D.G.P.Tc. M. T. Tc. C.P.L.T.Tc.

MANUAL

DE PROGRAMARE ÎN LIMBAJ DE ASAMBLARE PENTRU MICROPROCESOARELE 8080 ȘI 8085

Traducere din limba engleză

Pentru uz intern

București
1983

M. T. Tc.

D. G. P. Tc.

C. P. L. T. Tc.

MANUAL

pentru perfecționarea pregătirii profesionale a personalului tehnic de specialitate din domeniul telecomunicațiilor cu privire la cunoașterea microprocesorului 8080 și utilizarea acestuia în tehnica telecomunicațiilor.

Responsabili din partea unităților coordonatoare pentru programul privind comutația telefonică automată electronică;

D. G. P. Tc.

ing. Pătulea Matei

C. P. L.T. Tc.

ing. Tomescu Ion

București
1983

COORDONATOR : ing.TONESCU C.ION

TRADUCEREA

ing.Stan Dumitru - capitolele: 1,2,3,7
ing.Csutak Jstván - capitolele: 4,5,6

REVIZIA TEHNICA

ing.Suciu Gabriel
ing.Tomescu Ion
ing. Negruţi Adriana
ing.Trană Badiţă
ing.Mirea Adrian
ing.Stănescu Gheorghe

DACTILOGRAFIERE

Lingureanu Rodica

C U P R I N S

Pag.

1. Microprocesor şi limbajul de asamblare 5
   Introducere 5
   Ce este un asamblor 5
   Generalităţi despre hardware 8080/8085 8
   Memoria 9
   Registrele de lucru 11
   Indicatori de condiţie 15
   Stiva şi indicatorul de stivă 18
   Porturi de intrare ieşire . . . . 21$
   Setul de instrucţiuni 22
   Suma hardware/instrucţiune 28
   Diferenţele microprocesorului 8085 33

2. Conceptele limbajului de asamblare 34
   Introducere 34
   Formatul liniei sursă 34
   Cîmpul codului de operare 36
   Prezentarea datelor în complement de 2 40
   Simboluri şi tabele de simboluri 43
   Evaluarea expresiei în timpul asamblării 46

3. Setul de instrucţiuni 55

4. Pseudoinstrucţiuni,declaraţii 135
   Definirea simbolurilor 136
   Definirea datelor 137
   Rezervare de memorie 140
   Asamblare condiţionată 143
   Sfîrşitul asamblării 146
   Controlul numărului de adrese şi realocarea 147
   Pseudoinstrucţiuni utilizate pentru relocare 150

5. Macroinstrucţiuni ("macros") 158
   Introducere 158
   Folosirea "macro"-urilor 161
   Apelări macro 172

Pag.

Macro-uri nule 177
Exemple de macto 178

6. Tehnici de programare 185
Pseudo-subrutine cu tabele de ramificaţie 185
Transferul de date la subrutine 187
Multiplicare şi divizare software 191
Adunarea şi scăderea multibyte 196
Adunarea zecimală 198
Scăderea zecimală 201

7. Intreruperi 204
Generalităţi 204
Subrutine de întrerupere scrise 207

Anexa A. Sumarul instrucţiunilor 209
Anexa B. Sumarul instrucţiunilor asamblorului 215
Sumarul pseudoinstrucţiunilor 216
Pseudoinstrucţiuni macro 217
Instrucţiuni de realocare 217

Anexa C. Setul de caractere ASCII 219
Anexa D. Tabele cu conversia binar- zecimal - hexazecimal 220
Anexa E. Codurile în hexazecimal ale instrucţiunilor microprocesorului 8080 230

1. MICROPROCESORUL SI LIMBAJUL DE ASAMBLARE

Introducere

Aproape fiecare rînd al codului sursă din limbajul de asamblare al programului sursă este translatat direct într-o instrucţiune maşină pentru un anumit procesor. Deci programatorul în limbaju de asamblare trebuie să fie familiarizat atît cu limbajul de asamblare cît şi cu procesorul.

In prima parte a acestui capitol este descris asamblorul. In cea de a doua parte sînt descrise particularităţile microprocesorului 8080 din punct de vedere al programatorului.

Diferenţele de programare dintre microprocesoarele 8080 şi 8085 sînt relativ minore. Aceste diferenţe sînt descrise pe scurt la sfîrşitul acestui capitol.

Ce este un asamblor ?

Un asamblor este un instrument software - un program - destinat să simplifice sarcina de scriere a programelor de calculator. Cine a încercat să scrie un program de calculator,direct sub forma recunoscută de maşină adică în cod binar sau hexazecimal, va aprecia avantajele programării în limbaj simbolic de asamblare.

Codurile operaţiunilor în limbajul de asamblare (op - cedic) sînt uşor de memorat (MOV pentru instrucţiunea de deplasare JIM pentru instrucţiunea de salt etc.).

Se pot de asemenea,exprima simbolic adresele şi valorile conţinute în cîmpul operand al instrucţiunii. După ce au fost asignate aceste denumiri, ele pot fi folosite mnemonice (prescurtări) pentru instrucţiuni. De exemplu dacă în program trebuie să fie manipulată o dată, i se poate asigna denumirea simbolică DATE. Dacă programul conţine un set de instrucţiuni folosit periodic în timp, acest grup de instrucţiuni poate fi de -

numit TIMER.

Ce face asamblorul ?

Pentru a folosi asamblorul este nevoie în primul rînd de un program sursă. Programul sursă este format din instrucţiuni scrise în limbaj de asamblare. Aceste instrucţiuni sînt scrise folosindu-se prescurtări pentru codurile de operare şi etichete, după cum s-a descris mai înainte.

Programele sursă în limbaj de asamblare, trebuie să fie editate sub o formă accesibilă maşinii (programe obiect) cînd trec la asamblor.

Sistemul dezvoltat de INTEL cuprinde un text editor care va ajuta la păstrarea.programelor sursă în fişiere pe benzi perforate sau diskette. Utilizatorul poate apoi, oricînd,să treacă programul sursă din fişier la asamblor.

Editorul de texte este descris în : ISIS - II System User's Guide.

Programul asamblor îndeplineşte sarcina de translatare a codului simbolic în cod obiect (coduri maşină) care poate fi executate de microprocesoarele 8080 şi 8085.

Asamblorul poate prezenta la ieşire programul sub trei forme :

1. OBJECT FILE, conţinînd programul translatat în cod obiect,

2. LIST FILE, lista în care e imprimat codul sursă codul obiect generat de asamblor şi tabela de simboluri,

3. SYMBOL - CROSS - REFERENCE FILE, lista simbolurilor de referire înregistrate în program .

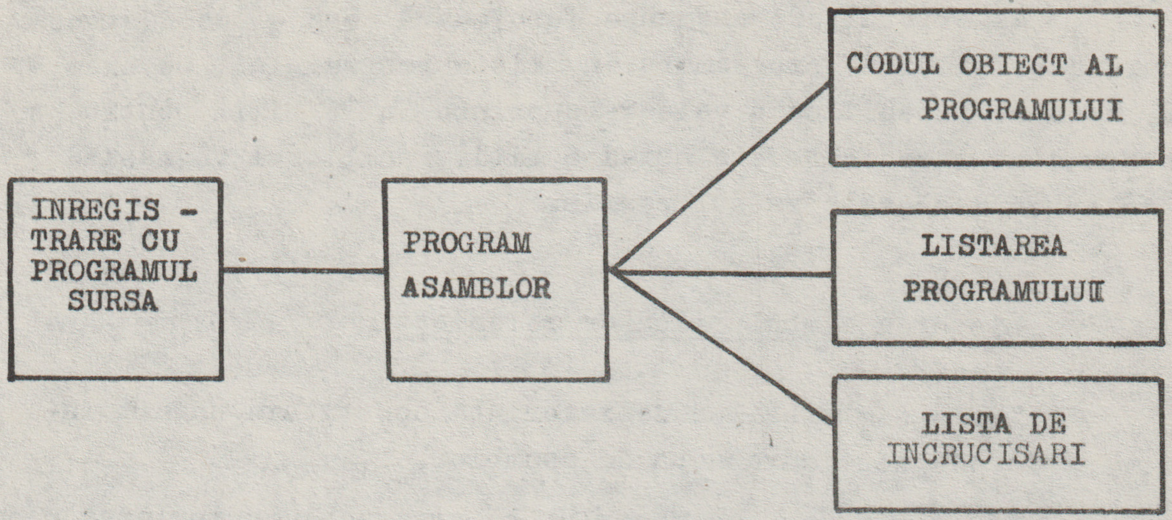


Fig.1-1. Reziltatele asamblării

Codul obiect

In majoritatea aplicaţiilor pe microcomputer, se va încărca programul obiect, sub diverse forme, într-o memorie ROM. Nu trebuie uitat că sistemul dezvoltat de INTEL este un microcompu - ter 8080, sistem cu memorie RAM.

In multe cazuri programul obiect poate fi încărcat şi executat pe un alt sistem dezvoltat pentru a fi testat şi depanat. Acesta permite testarea programului înainte ca sistemul să fie complet dezvoltat.

O trăsătură specială a asamblorului este aceea că el permite cererea codului obiect în format relocabil. Acesta eliberează programatorul de grija amestecării memoriilor ROM şi RAM în sistem. Există de asemenea posibilitatea, la nevoie, de relocare a unor porţiuni separate din program, dacă scopurile aplicaţiei o cer. De asemenea, un program mare poate fi descompus într-un număr de module asamblate separat. Fiecare modul este mai uşor de codificat şi testat. A se vedea capitolul 4 din acest manual în care sînt descrise pe larg avantajele facilităţilor de relocare (reamplasare).

Lista programului

Lista programului conţine înregistrarea permanentă atît a programului sursă cît şi a codului obiect.

Asamblorul, de asemenea furnizează mesaje de diagnosticare a erorilor de programare în lista programului. De exemplu, dacă se specifică o valoare cuprinsă în 16 biţi pentru o instrucţiune care foloseşte numai 8 biţi, asamblorul va sesiza că valoarea depăşeşte rangul permis.

Listarea simbolurilor de referinţă

Lista simbolurilor de referinţă constituie un alt instrument de diagnosticare adus de asamblor.

Să presupunem, de exemplu, că programul manipulează un cîmp de date numit simbolic DATE şi că testarea descoperă o eroare în logica programului de mînuire a acestor date. Lista simbolurilor de referinţă simplifică depanarea acestor erori, deoarece ea prezintă fiecare instrucţiune care face referire la simbolul DATE.

Generalităţi despre hardware 8080/8085

Pentru programator, computerul cuprinde următoarele părţi :

- memoria ;
- contorul de program ;
- registre de lucru ;
- flag-uri (indicatori) de condiţie ;
- stivă şi indicator de stivă ;
- porturi de intrare/ieşire;
- setul de instrucţiuni.

Dintre componentele listate mai sus, memoria nu face parte din procesor, dar prezintă interes pentru programator.

Memoria

Deoarece programul care dirijează microprocesorul se găseşte în memorie, toate aplicaţiile microprocesoarelor necesită memorii. Există două tipuri generale de memorii :

- memorii numai citire, ROM (Read Only Memory) ;
- memorii cu acces aleator, RAM, (Randon Acces Memory).

ROM

Aşa după cum reiese din denumirea memoriei, procesorul poate numai să citească instrucţiunile şi datele din ROM; el nu poate altera conţinutul acestor memorii.

Spre deosebire, într-o memorie RAM, procesorul poate atît să scrie (stocheze) cît şi să citească din ea.

Instrucţiunile şi datele neschimbate sînt fixate permanent în ROM şi rămîn intacte dacă alimentarea este sau nu conectată la sistem. Din acest motiv este folosită memoria ROM, pentru înmagazinarea programului. Cu memorie ROM, există certitudinea că programul este gata pentru execuţie după ce sistemul a fost alimentat. In cazul memoriei RAM, programul trebuie să fie încărcat din nou în memorie de fiecare dată cînd alimentarea este conectată la procesor. De notat, printre altele că sistemul este multiscop, deoarece în memoria RAM pot fi încărcate diverse programe pentru diferite aplicaţii.

Două alte tipuri de memorii ROM - PROM (Programmable Read Only Memory) şi EPROM (Erasamble Programmable Read Only Memory) sînt folosite frexvent în timpul dezvoltării programului.

Aceste memorii sînt folosite în timpul dezvoltării programului deoarece ele pot fi schimbate de un programator

special PROM.

RAM

Chiar dacă programul este conţinut integral într-o memorie ROM, este posibil ca aplicaţiile să necesite o memorie RAM. Cîteodată programul trebuie să înscrie date în memorie, care în acest caz trebuie să fie RAM.

Dacă programul utilizează stiva, atunci va fi necesar un RAM.

O combinaţie de memorii ROM şi RAM este importantă în aplicaţii pentru proiectantul de sistem şi pentru programator.

Normal,programatorul trebuie să cunoască adresele fizice ale memoriei RAM din sistem, unde pot fi asignate datele variabile la aceste adrese.

Contorul de program

Pentru contorul de program este luat primul din registrele interne ale microprocesorului 8080, ilustrat în figura 1 - 3.

NOTA : cu excepţia diferenţelor listate la sfîrşitul acestui capitol, informaţiile din acest capitol se folosesc atît la 8080 cît şi la 8085.

Contorul de program indică locul următorului bait din instrucţiunea ce urmează să fie extrasă din memorie (care poate fi ROM sau RAM).

De fiecare dată,după ce baitul instrucţiunii este extras din memorie, procesorul încrementează cu 1 contorul de program. Controul de program întotdeauna indică următorul bait ce trebuie extras. Acest proces continuă în tot cursul programului pe măsură ce instrucţiunile sînt executate rînd pe rînd (secvenţial).

Pentru a se schimba fluxul de execuţie al programului cu o instrucţiune JUMP sau CALL către o subrutină, procesarul înlocuieşte conţinutul curent al contorului de program cu adresa noii instrucţiuni. Următoarea instrucţiune este apoi extrasă din noua adresă.

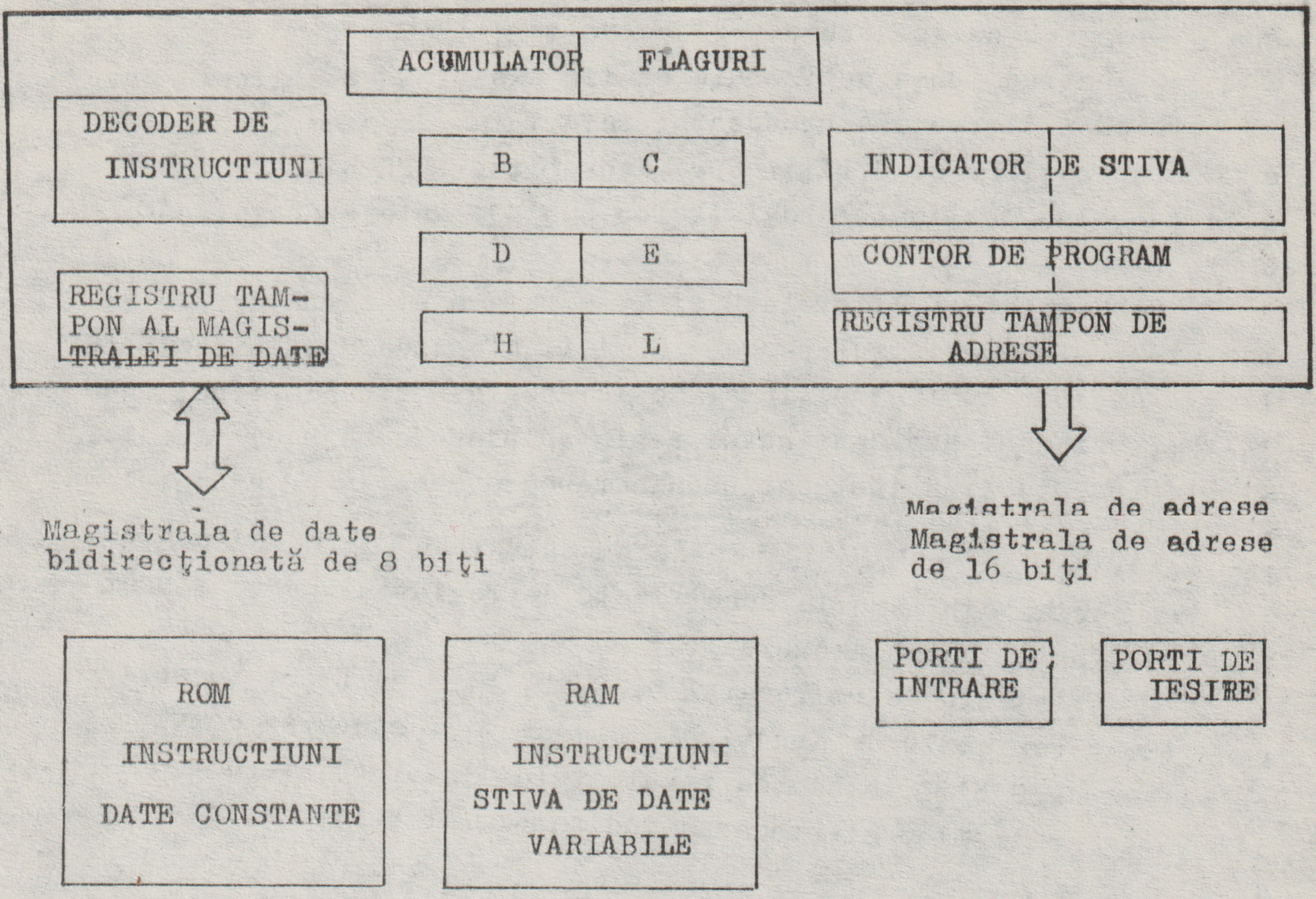


Fig.1.3. Registre interne 8080/8085

<u>Registrele de lucru</u>

Microprocesorul 8080 este prevăzut cu un registru acumulator cu 8 biţi şi alte 6 registre de lucru de uz general, după cum este arătat în figura 1.3.

Programul face referiri la aceste registre prin literele A (pentru acumulator), B,C,D,E,H şi L. Astfel instrucţiunea ADD B poate fi interpretată ca : adună conţinutul registrului B la conţinutul acumulatorului.

O serie de instrucţiuni pot face referiri la perechi de registre după cum urmează :

Simbol de referinţă	Registru pereche indicat
B	B şi C
D	D şi E
H	H şi L
M	H şi L (referinţă la memorie)
PSW	A şi (indicaţii de condiţie (explicaţii ulterioare în acest capitol)

Simbolul de referinţă pentru un singur registru este adesea acelaşi cu cel pentru o pereche de registre.

Instrucţiunea ce trebuie să fie exefutată stabileşte cum trebuie să interpreteze procesorul referinţa. De exemplu, ADD B este o operaţiune cu 8 biţi. Spre deosebire, PUSH B (care introduce în stivă conţinutul registrelor B şi C) este o operaţiune cu 16 biţi.

De notat că literele H şi M, ambele sînt simboluri de referinţă pentru registrul pereche H şi L. Alegerea modului de utilizare depinde de instrucţiune. Se foloseşte H cînd instrucţiunea INX H (încrementează conţinutul registrului pereche H şi L cu 1). M se foloseşte cînd instrucţiunea adresează memoria cu ajutorul registrelor pereche H şi L , ca de exemplu : ADD M (adună conţinutul locaţiei din memorie specificată de registrul pereche H şi L la conţinutul acumulatorului).

Registrele de uz general B, C, D, E, H şi L pot îndeplini o largă varietate de funcţiuni, cum ar fi : stocarea datelor cu valori de 8 biţi, stocarea rezultatelor intermediare din operaţiunile aritmetice şi stocarea indicatorilor pentru adresele de 16 biţi.

Deoarece setul de instrucţiuni 8080 este foarte variat , este posibil să se obţină acelaşi rezultat de operare cu diferite instrucţiuni. De exemplu, o simpă adunare la acumulator, poate fi obţinută cu mai mult de jumătate de duzină de instrucţiuni diferite. Este posibil, şi în general este de dorit, să se aleagă o instrucţiune registru la registru cum ar fi ADD B.

Această instrucţiune, de obicei, necesită numai un bait în programul înmagazinat. De asemenea, folosindu-se date deja prezente în registre, se elimină accesul la memorie şi astfel se reduce timpul necesar de operare.

Acumulatorul, de asemenea, poate fi folosit ca registru de uz general, dar el are o serie de posibilităţi speciale pe care celelalte registre nu le au. De exemplj, instrucţiunile de intrare/ieşire IN şi OUT transferă date numai între acumulator şi perifericele I/O.

Exemple:

In următoarele două figuri este ilustrată executarea unei instrucţiuni MOV.

Instrucţiunea MOV M,C transferă o copie a conţinutului registrului C în locaţia din memorie care a fost indicată de registrele H şi L. De notat că această locaţie trebuie să fie într-o memorie RAM deoarece datele trebuiesc înscrise în memorie.

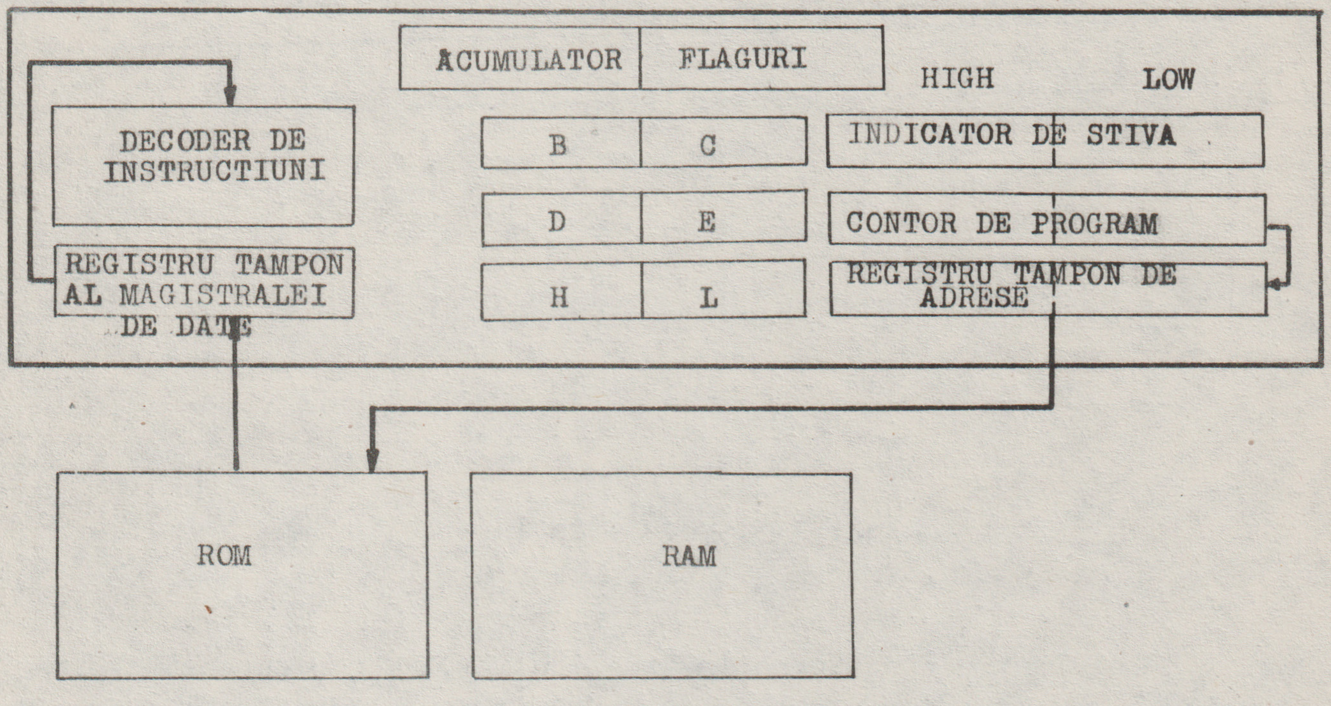


Fig.1-4 Aducerea instrucţiunii pentru decodificare

Procesorul iniţiază aducerea instrucţiunii prin introducerea conţinutului contorului de program pe busul de adresare şi apoi incrementează cu 1 contorul de program pentru a indica adresa baitului următoarei instrucţiuni. Cînd memoria răspunde, instrucţiunea este decodată într-o serie de acţiuni după cum este arătat în figura 1 - 5.

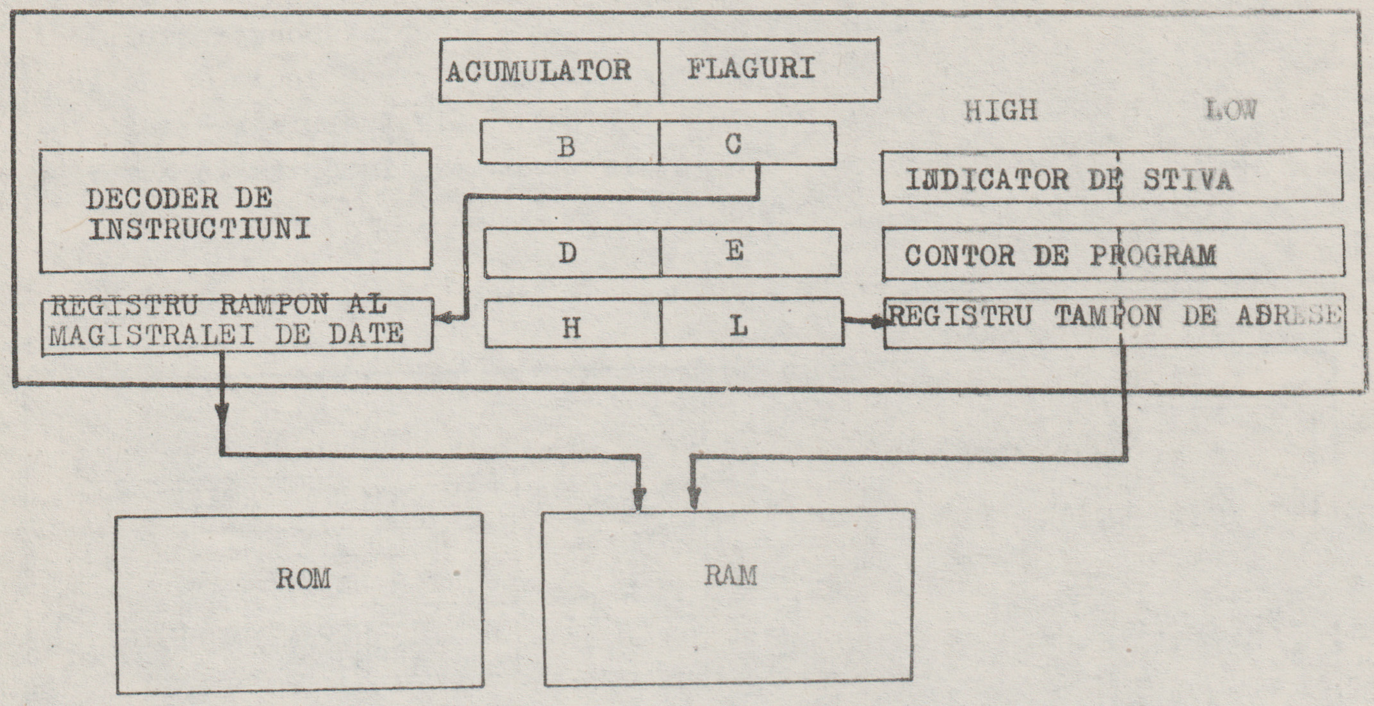


Fig. 1-5 Executarea instrucţiunii
MOV M,C

Pentru executarea instrucţiunii MOV M,C procesorul introduce conţinutul registrului C pe busul de date şi conţinutul registrelor H şi L pe busul de adrese.

Cînd memoria acceptă datele, procesorul termină execuţia acestei instrucţiuni şi iniţiază extragerea următoarei instrucţiuni.

Registrele interne de lucru

Anumite operaţiuni sînt distructive. De exemplu, o comparaţie este de fapt o operaţie de scădere ; un rezultat zero indică egalitatea dintre doi operanzi.

Deoarece e inacceptabil să fie distrus unul din operanzi, procesorul conţine mai multe registre de lucru rezervate acestui scop.

Programatorul nu poate avea acces la aceste registre.

Aceste registre sînt folosite pentru ransferul intern al datelor şi conservarea operanzilor în cazul operaţiunilor distructive.

Indicatorii de condiţie

Microprocesorul 8080 este prevăzut cu 5 bistabile folosite pentru indicatorii de condiţie (FLAG FLIP - FLOPS) care au următoarea semnificaţie : zero (Z), semn (S), paritate (P), transport (CY) şi transport auxiliar (AC).

Anumite instrucţiuni aritmetice şi locigee pot schimba unul sau mai mulţi din aceşti indicatori pentru a indica rezultatul operaţiei.

Programul poate testa starea a patru din aceşti indicatori (transport, semn, zero şi paritate) folosind una din instrucţiunile : salt condiţionat, call return. Aceasta permite schimbarea cursului executării programului bazat pe rezultatul operaţiei anterioare.

Al cincilea indicator, transportul auxiliar (AC) este rezervat pentru a fi folosit cu instrucţiunea DAA (pentru schimbarea în zecimal a conţinutului acumulatorului), care va fi explicată ulterior în acest capitol.

Este important pentru programator să ştie ce indicator este modificat de o anumită instrucţiune.

Să presupunem, de exemplu, că programul testează paritatea unui bait la intrare şi apoi execută o servenţă de instrucţiuni dacă paritatea este pară sau un alt set de instrucţiuni dacă partitatea este impară.

Codificînd o instrucţiune JPE (salt dacă paritatea este pară) sau JPO (salt dacă paritatea este impară), imediat după instrucţiunea IN (de intrare), acestea vor produce rezultate false deoarece instrucţiunea IN nu afectează indicatorii de condiţie.

Pentru ca operaţiunea să se desfăşoare corect trebuie să se includă nişte instrucţiuni care să schimbe indicatorul paritate după instrucţiunea IN, dar înaintea instrucţiunilor de salt JPE sau JPO.

De exemplu, se poate aduna zero la acumulator. Aceasta setează indicatorul paritate fără schimbarea datelor în acumulator.

In alte cazuri, dacă se doreşte schimbarea unui indicator cu o instrucţiune, dar există alte instrucţiuni care intervin mai înainte, se pot folosi acestea. In aceste cazuri, trebuie să existe certitudinea că, intervenţia instrucţiunilor

nu a afectat indicatorul dorit.

Setul de indicatori pentru fiecare instrucţiune este prezentat detaliat în capitolul 3 care cuprinse setul complet de instrucţiuni 8080.

Notă : Cînd un indicator este "set" sau ON,(are valoarea 1) ;
Cînd un indicator este "reset"sau OFF,(are valoarea 0).

Indicatorul de transport CY

Aşa cu, reiese din denumire, indicatorul CY este de regulă folosit pentru a indica dacă dintr-o adunare rezultă un "transport" spre următorul digit de ordin superior.

Indicatorul CY este de asemenea folosit ca indicator de "împrumut" la o scădere, după cum este explicat în capitolul 2, despre date reprezentate în complement.de doi.

Indicatorul CY este afectat de instrucţiunile logice AND, OR şi OR EXCLUSIV. Aceste instrucţiuni pun ON sau OFF anumiţi biţi din acumulator. A se vedea descrirea instrucţiunilor ANA, ANI, ORA, ORI, XRA şi XRI în capitolul 3.

Instrucţiunile de rotire, care deplasează conţinutul acumulatorului cu o poziţie spre stînga sau dreapta tratează bitul de transport (CY) ca şi cum ar fi cel de al nouălea bit al acumulatorului. A se vedea descrierea instrucţiunilor RAL, RAR, RIC şi RRC în capitolul 3.

Exemplu :
Adunarea a două numere de un bait(octet) poate produce un transport spre bitul de ordin superior.

Numărul bitului :		7 6 5 4	3 2 1 0
Valoarea în hexazecimal :	AE =	1 0 1 0	1 1 1 0
Valoarea în hexazecimal :	+ 74 =	0 1 1 1	0 1 0 0
		0 0 1 0	0 0 1 0 =

= 22 H şi CY = 1

La operaţia de adunare din care rezultă un transport bitul de ordin superior, indicatorul CY este trecut în satreal, iar la adunarea din care nu rezultă un transport, CY = 0.

Indicatorul de semn, S

In capitolul 2, despre reprezentarea în complementul lui 2 a datelor, s-a explicat că bitul 7 al rezultatului din cumulator poate fi interpretat ca semn.

Instrucţiunile care afectează indicatorul de semn S face bitul S egal cu bitul 7.

Un zero în bitul 7 indică o valoare pozitivă,iar un unu indicaă o valoare negativă. Această valoare este duplicată în indicatorul S aşa încît instrucţiunile salt condiţionat CALL şi RETURN pot testa valori pozitive şi negative.

Indicatorul de paritate P

Paritatea este determinată prin numărarea bitilor de valoare 1 conţinuţi în acumulator.

Instrucţiunile care afectează indicatorul de paritate pun P = 1 pentru paritate pară şi P = 0 pentru paritatea impară.

Indicatorul de zero Z

Anumite instrucţiuni modifică indicatorul de zero Z dîndu-i valoarea 1 pentru a evidenţia că toţi biţii din acumulator au valoarea zero. Aceste instrucţiuni fac Z = 0 dacă rezultatul din acumulator are o valoare diferită de zero. Un exemplu de la care se obţine un transport, CY = 1 şi un bait cu toţi biţii zero, Z = 1, este arătat mai jos :

```
  1010 0111
+ 0101 1001
-----------
1 0000 0000    CY = 1
```

Indicatorul de transport auxiliar AC

Flagul de transport auxiliar indică un transport rezultat din bitul 3 al acumulatorului.

Acest indicator nu poate fi testat direct în program;el este folosit pentru a permite executarea instrucţiunii DAA (Decimal Adjust Accumulator).

Indicatorul de transport auxiliar şi instrucţiunea DAA tratează valoarea conţinută în acumulator ca pe două cifre ze -

cimale codate binar cu cîte 4 biţi fiecare. Astfel valoarea 0001 1001 este echivalentă cu 19 (în zecimal). Dacă aceastâ valoare este întreruptă ca un număr binar el are valoarea 25. (11001B = 25 D)

B = binar

D = zecimal

De notat că, oricum dacă se adună la această valoare se produce un rezultat nezecimal :

0001 1001
0000 0001

0001 1010 = 1 A în hexazecimal

Instrucţiunea DAA converteşte valorile hexazecimale cum ar fi A din exemplul precedent,înapoi în format zecimal codat binar (BCD).

Instrucţiunea DAA cere indicator de transport auxiliar deoarece formatul BCD face posibile operaţiunile aritmetice care generează un transport de la digitul de 4 biţi de ordin inferior spre digitul de 4 biţi de ordin superior.

Instrucţiunea DAA execută următoarele adunări pentru a corecta rezultatul din exemplu precedent :

0001 1010
\+ 0000 0110

0001 0000
0001 0000 (transport auxiliar

0010 000 = 20 în zecimal = 19 + 1

Indicatorul de transport auxiliar AC este afectat de toate instrucţiunile de adunare, scădere,incrementare, decrementare,comparare şi de operaţiile logice SI, SAU şi SAU EXCLUSIV. (A se vedea descrierea acestor instrucţiuni în cap.3).

<u>Stiva şi indicatorul de stivă</u>

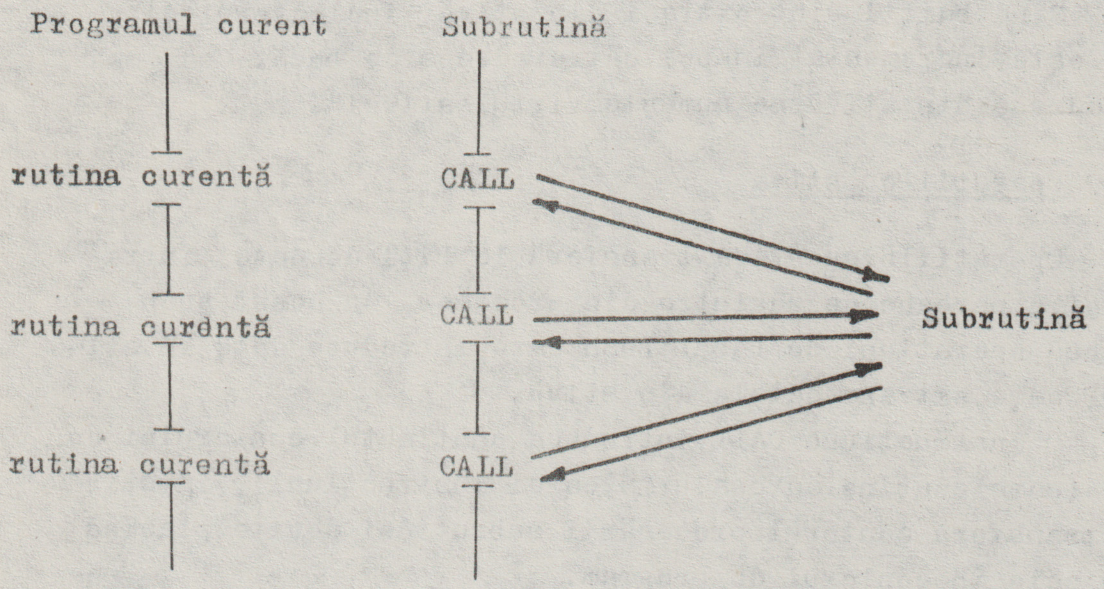
Pentru a înţelege scopul şi eficacitatea stivei este necesar mai întîi să fie înţeles conceptul de subrutină.

Să presupunem că programul necesită o rutină de multiplicare. (Deoarece microprocesorul 8080 nu are instrucţiuni de multiplicare aceasta poate fi realizată prin adunare repetată.

De exemplu, 3 x 4 este echivalent cu 3+3+3+3).

In cele ce urmează se presupune că programul are nevoie de această rutină de multiplicare de mai multe ori. Această rutină poate fi înregistrată de atîtea ori cît este necesar să fie folosită însă în acest fel se ocupă un spaţiu mare în memorie.

O subrutină se poate coda:



Microprocesorul 8080 este prevăzut cu instrucţiuni de chemare şi întoarcere dintr-o subrutină. Cîns este executată o instrucţiune CALL, adresa următoarei instrucţiuni (conţinută în contorul de program) este introdusă în stivă. Conţinutul contorului de program este înlocuit cu adresa subrutinei dorite.

La terminarea subrutinei, o instrucţiune de întoarcere extrage din stivă adresa introdusă anterior şi o reîncarcă apoi în contorul de program, după care execuţia programului continuă.

Mecanismul care face posibilă această funcţionare este stiva. Stiva este o zonă rezervată în memoria RAM care poate fi adresată cu ajutorul indicatorului de stivă SP (Stack Pointer).

Indicatorul de stivă este un registru hordware controlat de procesor. In orice caz, programul trebuie să iniţializeze indicatorul de stivă; aceasta înseamnă că programul tre -

buie să încarce adresa de bază a stivei în indicatorul de sti - vă SP.

Adresa de bază a stivei este de obicei asigurată la adresa disponibilă cea mai de sus din memoria RAM. Aceasta, din cauză că extinderea stivei se face prin decrementarea indicatorului de stivă. Baiţii adăugaţi în stivă, se extind în locaţiile de memorie cu adrese inferioare.

Cînd baiţii sînt extraşi din stivă, indicatorul de stivă SP este incrementat înapoi către adresa de bază. Cea mai recentă adresă din stivă se numeşte"vîrful stivei".

Operaţii cu stiva

Operaţiile cu stiva transferă 16 biţi de date între memorie şi o pereche de registre din procesor. In acest scop există două operaţiuni de bază: PUSH care introduce date în stivă şi POP care extrage datele din stivă.

O instrucţiune CALL introduce conţinutul contorului de program, (care conţine adresa următoarei instrucţiuni), în stivă şi apoi transferă contorul procesării subrutinei dorite plasînd adresele sale în contorul de program.

Instrucţiunea POP de revenire la programul de bază,extrage 16 biţi din stivă şi apoi îl plasează în contorul de pro - gram. In aceste condiţii programatorul trebuie să ştie ce se găseşte în stivă. De exemplu, dacă se apelează o subrutină şi subrutina introduce date în stivă, subrutina trebuie apoi să re - transfere aceste date înainte de executarea instrucţiunii de revenire. In caz contrar, instrucţiunea de revenire extrage date din stivă şi le va plasa în contorul de program.

Rezultatul e imprevizibil şi probabil diferit de ce s-a dorit.

Salvarea stării programului

Exist posibilitatea ca o subrutină să necesite folo - sirea unuia sau a mai multor registre de lucru. Printre altele, de asemenea este posibil ca programul de bază să aibă date în prelucrare, conţinute în aceste registre, date care nu trebuie să fie alterate de subrutină cînd aceasta va folosi registrele de lucru. Pentru a se evita alterarea datelor, subrutina tre -

buie să salveze conţinutul registrelor înainte de a le folosi şi să refacă acest conţinut înainte ca programul de bată să preia controlul.

Subrutina poate executa această sarcină prin transferarea datelor din registre în stivă şi apoi extragerea din stivă şi readucerea acestor date din nou în registre, înainte ca instrucţiunea de revenire la programul de bază să fie executată.

Secvenţa următoare de instrucţiuni salvează şi restabileşte conţinutul tuturor registrelor de lucru.

De notat că instrucţiubea POP trebuie să fie în ordine opusă cu instrucţiunea PUSH dacă datele sînt restocate în locaţiile lor avute anterior.

```
SUBRTN: PUSH  PSW
        PUSH  B
        PUSH  D
        PUSH  H

        Subrutina

        POP  H
        POP  D
        POP  B
        POP  PSW
        RETURN
```

Listele B, D şi H se referă la registrele pereche B şi C, D şi E, H şi L Registrul PSW (Program Status Word) conţine cuvîntul de stare a programului.

Cuvîntul de stare a programului are 16 biţi care conţin acumulatorul şi cei 5 indicatori de condiţie.

<u>Porturi de intrare/ieşire</u>

Sînt prevăzute 256 porturi de intrare/ieşire pentru comunicaţiile cu toate dispozitivele periferice. Transferul de date între microprocesor şi periferice este iniţiat de instrucţiunile IN şi OUT.

Indtrucţiunea IN introduce pe magistrala (busul) de adrese numărul portului dorit. Imediat ce un bait de date apare pe busul de date, el va fi transferat în acumulator.

Instrucţiunea OUT introduce pe magistrala de adrese numărul portului dorit. De asemenea, din acumulator prin interfaţa busului de date este transmis un bait de date către periferic.

Numărul portului specificat este duplicat pe magistrala de adrese. Astfel, instrucţiunea IN 5 introduce pe magistrala de adrese următoarea secvenţă de biţi. 0000 0101 0000 0101.

De notat că instrucţiunile IN SI OUT, pur şi simplu nu fac decît un transfer de date. Sarcina de a detecta dacă perifericele au fost corect adresare rămîne în seama acestora. De notat, de asemenea, că numărul de adresare a dispozitivului poate fi schimbat.

Pentru detalii se vor vedea instrucţiunie IN, OUT, DI, EI, RST,RIM şi SIM din capitolul 3. (RIM şi SIM numai pentru 8085).

<u>Setul de instrucţiuni</u>

Microprocesorul 8080 include un set puternic de instrucţiuni. In această secţiune setul de instrucţiuni este prezentat în mod general. Detalii asupra fiecărei instrucţiuni sînt date în capitolul 3.

<u>Moduri de adresare</u>

Instrucţiunile pot fi împărţite în diverse categorii în funcţie de modul de adresare a registrelor şi/sau memoriei.

<u>Adresare implicită.</u> Modul de adresare al anumitor instrucţiuni este implicat de funcţia pe care trebuie să o îndeplinească instrucţiunea respectivă. De exemplu, instrucţiunea STC (modifică transportul : CY = 1) are sarcina precisă de modificare a indicatorului CY ; instrucţiunea DAA (schimbă valoarea acumulatorului în zecimal), are sarcină precisă asupra acumulatorului.

<u>Adresare registru</u>

Foarte multe instrucţiuni care utilizează registrele pentru executarea operaţiunilor, folosesc adresarea de registru. In aceste instrucţiuni se poate specifica unul din registrele A la E, H sau L , corespunzător codului de operare. In cazul aces-

tor instrucţiuni acumulatorul (registrul A) este considerat în mod automat ca fiind cel de al doilea operand. De exemplu, instrucţiunea CMP E poate interpreta în felul următor: compară conţinutul registrului E cu conţinutul acumulatorului,

O mare parte din instrucţiuni folosesc adresarea registru pentru operaţiuni cu valori de 8 biţi. Există cîteva ins - trucţiuni care execută operaţii cu 16 biţi în care caz sînt folosite registre pereche (B şi C, D şi E, H şi L). De exemplu , instrucţiunea PCHL schimbă cei 16 biţi din contorul de program PC, cu conţinutul registrelor pereche H şi L.

Adresare imediată

Instrucţiunile care folosesc adresarea imediată cuprind datele asamblate într-o parte a instrucţiunii. De exemplu, ins_ trucţiunea CPI'C 'este interpretată în felul următor : compară conţinutul acumulatorului cu litera C din codul ASCII (C=43H= 0100 0011 B).

Cînd este asamblată această instrucţiune are valoarea hexazecimală FE 43.(codul de operare CPI=FEH= 1111 1110 B).

Cînd este executată această instrucţiune procesorul extrage primul baiţ al instrucţiunii şi determină dacă trebuie să aducă mai mult de un bait. Procesorul extrage următorul bait al instrucţiunii şi îl plasează într-unul din registre după care îndeplineşte operaţia de comparare.

De notat că denumirea de instrucţiune imediată evidenţiază cazurile cînd datele sînt conţinute în instrucţiune şi pot fi folosite imediat.

Codul de operare al unei astfel de instrucţiuni se termină cu litera I, astfel instrucţiunea de adunare imediată ADI se deosebeşte de instrucţiunea de adunare registru/memorie ADD.

In ambele cazuri, în care primul operand fie că este conţinut în instrucţiune (imediat), fie că se găseşte într- un registru sau memorie cel de al doilea operand poate fi conţi - nut în acumulator sau registru/memorie.

Exemplu : instrucţiunea MVI poate transfera imediat datele specificate în instrucţiune, în oricare din registrele de lucru, inclusiv în acumulator sau în memorie. Astfel instrucţiunea MVI D, OFFH transferă valoarea hexazecimală FF în regis-

trul D. Instrucţiunea LXI (încarcă imediat registrul pereche) se foloseşte pentru încărcarea imediată a datelor cu valori de 16 biţi.

Această instrucţiune este frecvent folosită pentru încărcarea adreselor într-un registru pereche. După cum s-a arătat anterior, programul trebuie să iniţializeze indicatorul de stivă SP. Instrucţiunea LXI este cea mai folosită în acest scop. De exemplu: instrucţiunea LXI SP, 30 FFH încarcă indicatorul de stivă SP cu valoarea hexazecimală 30 FF, indicată imediat în instrucţiune.

Adresare directă

Instrucţiunile de salt (JUMP) au nevoie pentru operare de adrese cu 16 biţi care sînt cuprinse chiar în instrucţiune făcînd parte din aceasta. De exemplu, instrucţiunea JMP 1000 H provoacă un salt la adresa hexazecimală 1000 prin înlocuirea conţinutului curent al contorului de program (PC) cu noua valoare 1000.

Instrucţiunile care cuprind adrese directe cer pentru stocare 3 baiţi ; un bait pentru codul de operare al instrucţiunii şi doi baiţi pentru adresa de 16 biţi.

Adresare indirectă prin registru

Instrucţiunile cu adresarea indirectă prin registru fac referinţe la memorie prin intermediul unor registre pereche. Astfel, instrucţiunea MOV M,C transferă conţinutul registrului C în memorie la adresa stocată în registrul pereche H şi L.

Instrucţiunea LDAX B încarcă acumulatorul cu baitul de date specificat de adresa conţinută în registrul pereche B şi C.

Mod de adresare combinat

O serie de instrucţiuni folosesc combinaţii ale modurilor de adresare. O instrucţiune CALL, de exemplu, combină adresarea directă şi adresarea indirectă prin registru. Adresa directă într-o instrucţiune CALL specifică adresa subrutinei dorite şi adresarea indirectă este indicatorul de stivă (SP).

Instrucţiunea CALL introduce conţinutul curent al contorului de program (PC) în memorie la locaţia specificată de indicatorul de stivă (SP).

Convenţii de numire a instrucţiunilor

Prescurtările asignate instrucţiunilor sînt stabilite , în aşa fel, ca să indice funcţia instrucţiunii respective.

Instrucţiunile pot fi cuprinse într-una din următoarele categorii funcţionale :

Grupa transferurilor de date

Instrucţiunile care execută transferuri de date deplasează date între registre sau între meorie şi registre.

MOV	transferă (mută)
MVI	transferă imediat
LDA	încarcă acumulatorul direct din memorie
STA	st chează acumulatorul direct în memorie
LHLD	încarcă registrul H şi L direct din memorie
SHLD	stochează registrul H şi L direct în memorie

Introducerea unui "X" în denumirea instrucţiunilor de transfer arată că transferul se face cu un registru pereche :

LXI	încarcă un registru pereche cu date imediate
LDAX	încarcă acumulatorul de la adresa dată de registrul pereche
STAX	stochează acumulatorul la adresa dată în registrul pereche
XCHG	schimbă H şi L cu D şi E
XTHL	schimbă vîrful stivei cu H şi L

Grupa aritmetică

Din această grupă fac parte instrucţiunile pentru adunarea, scăderea, incrementarea sau decrementarea datelor din registra sau memorie.

ADD	adună la acumulator
ADI	adună datele imediate la acumulator
ADC	adună la acumulator folosind trabsportul C
ACI	adună datele imediate la acumulator folosind transportul C
SUB	scade din acumulator
SUI	scade datele imediate din acumulator
SBB	scade din acumulator folosind împrumutul C

SBI scade din acumulator datele imediate folosind îm - prumutul C

INR incrementează cu 1 baitul specificat

DCR decrementează cu 1 baitul specificat

INX incrementează cu 1 registrul pereche

DCX decrementează cu 1 registrul pereche

DAD adună registre pereche: adună conţinutul unui registru pereche cu registrul pereche H şi L.

Grupa logică

Instrucţiunile din această grupă execută operaţiile logice (Boolean) cu datele din registre, memorie şi din indicatorii de condiţie.

Instrucţiunile logice AND, OR şi OR exclusiv (SI, SAU şi SAU exclusiv) permit ca anumiţi biţi din acumulator să fie trecuţi în stare unu sau zero.

ANA AND logic cu acumulatorul

ANI AND logic cu acumulatorul folosind date imediate

ORA OR logic cu acumulatorul

ORI OR logic cu acumulatorul folosind date imediate

XRA OR exclusiv logic cu acumulatorul

XRI OR exckusiv logic folosind date imediate

Instrucţiunule de comparare compară conţinutul unei valori de 8 biţi cu conţinutul acumulatorului.

CAM compară

CPI compară datele imediate

Instrucţiunile de rotire deplasează cu o poziţie de bit acumulatorul spre stînga sau spre dreapta.

RLC rotire stînga acumulator

RRC rotire dreapta acumulator

RAL rotire stînga prin CY

RAR rotire dreapta prin CY

Instrucţiuni de complementare a acumulatorului şi instrucţiuni ale bitului CY

CMA complementează acumulatorul

CMC completează transportul CY

STC schimbă transportul în CY = 1

Grupa de ramificaţie (de salt)

Instrucţiunile de ramificaţie schimbă fluxul secvenţial al programului necondiţionat sau condiţionat după cum se ţine sau se ţine seama de indicatorii de condiţie.

Instrucţiunile de salt necondiţionat sînt următoarele:

JMP salt
CALL apel
RET retur

Instrucţiunile de salt condiţionat examinează starea unuia din cei 4 indicatori de condiţie pentru a stabili dacă saltul trebuie să fie executat. Condiţiile de care trebuie să se ţină seama sînt :

NZ	Nu este zero (Z=0)
Z	Zero (Z=1)
NC	Nu este transport (C=0)
C	Transport (C=1)
PO	Paritate impară (P=0)
PE	Partitate pară (P=1)
P	Plus (S=0)
M	Minus (S=1)

Instrucţiunile de salt condiţionat sînt următoarele :

Salturile	Apeluri	Return-uri	
JC	CC	RC	(Transport)
JNC	CNC	RNC	(Nu este transport)
JZ	CZ	RZ	(Zero)
JNZ	CNZ	RNZ	(Nu este zero)
JP	CP	RP	(Plus)
JM	CM	RM	(Minus) ıs)
JPE	CPE	RPE	(Paritate pară)
JPO	CPO	RPO	(Paritate impară)

Două alte instrucţiuni pot efectua o ramificare pentru înlocuirea conţinutului contorului de program PC :

PCHL transferă H şi L în PC
RST instrucţiune specială pentru restart utilizată pentru întreruperi

Instrucţiunile de stivă, I/O şi control maşină. Următoarele instrucţiuni afectează stiva şi/sau indicatorul de stivă :

PUSH	introduce doi biţi de date în stivă
POP	extrage doi biţi de date din stivă
XTHL	schimbă vîrful stivei cu H şi L
SPHL	transferă conţinutul lui H şi L în SP (Stack Pointer)

Instrucţiunile de I/O sînt următoarele :

IN	iniţiază o operaţie de intrare
OUT	iniţiază o operaţie de ieşire

Instrucţiunile de control maşină sînt următoarele :

EI	activează sistemul de întreruperi
DI	dezactivează sistemul de întreruperi
HLT	oprire (Halt)
NOP	instrucţiune neoperativă

Sumar hardware/instrucţiune

In următoarele ilustraţii grafice este arătată funcţionalitatea hardware a micropro esorului în funcţie de grupele de instrucţiuni. Tipul operandului permis fiecărei instrucţiune este indicat prin folosirea unui cod. Cînd nu este dat nici-un cod instrucţiunea nu are operand.

Codul	Semnificaţia
$REGM_8$	Operandul specifică unul din registrele de 8 biţi A,B,C,D,E,H,L sau memoria M (referirea la memorie se face prin adrese de 16 biţi conţinute în registrul H şi L)
D_8	Desemnează un operand imediat de 8 biţi
A_{16}	Desemnează o adresă de 16 biţi
P_8	Desemnează numere de periferice de 8 biţi
REG_{16}	Desemnează registru pereche de 16 biţi (B şi C,D şi E,H şi L sau SP)

D_{16} Desemnează un operand imediat de 16 biţi .

Instrucţiunile ce operează cu acumulatorul

In figura următoare sînt arătate instrucţiunile care au efect asupra acumulatorului. Instrucţiunile care sînt listate deasupra acumulatorului acţionează asupra datelor din acumulator, şi toate cu excepţia CAM (complementează acumulatorul) , au efect şi asupra uneia sau mai multor indicatori de condiţie.

Instrucţiunile listate sub acumulator transferă date în sau din acumulator, dar nu afectează indicatorii de condiţie.

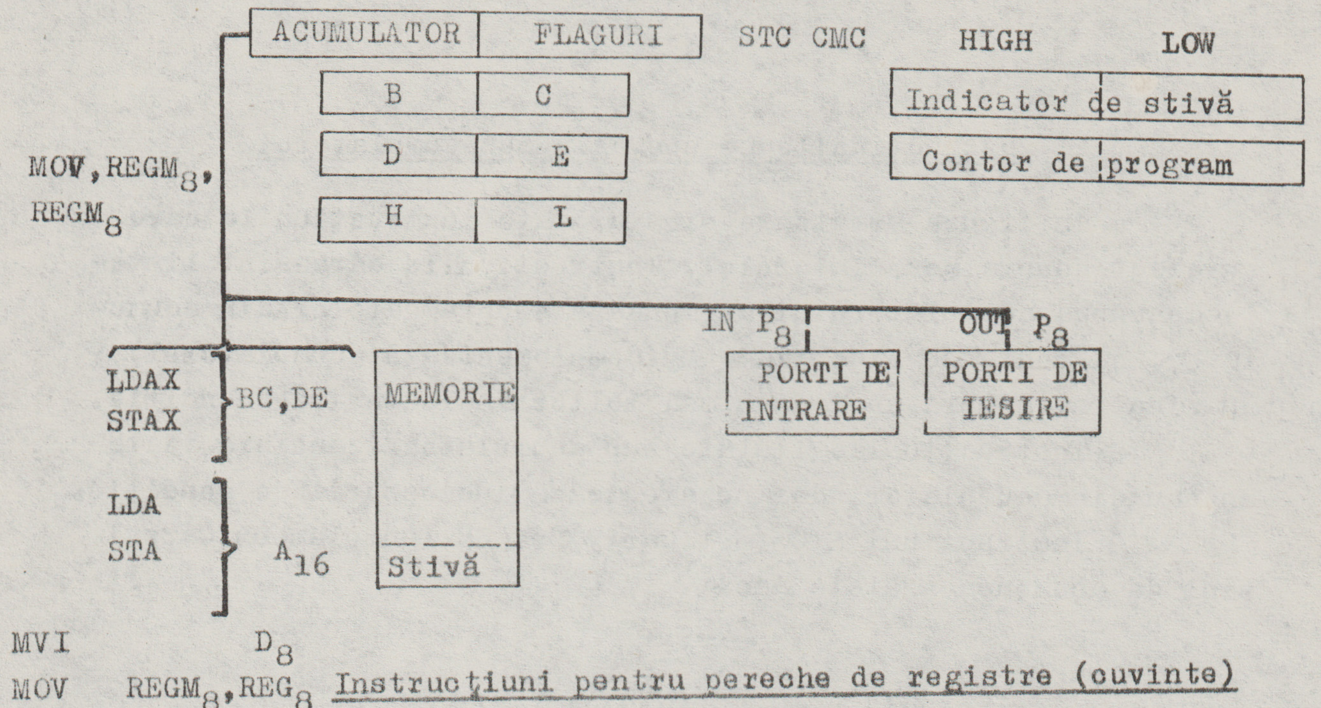
Instrucţiuni STC (Set Carry) şi CM (Complement Carry) sînt de asemenea arătate aici .

ADD, ABC, SUB, SBB, ANA, XRA, ORA, CMP	$REGM_8$	ADI, ACI, SUI, SBI, ANI, XRI, ORI, CPI	D_8

RLC RAL RRC

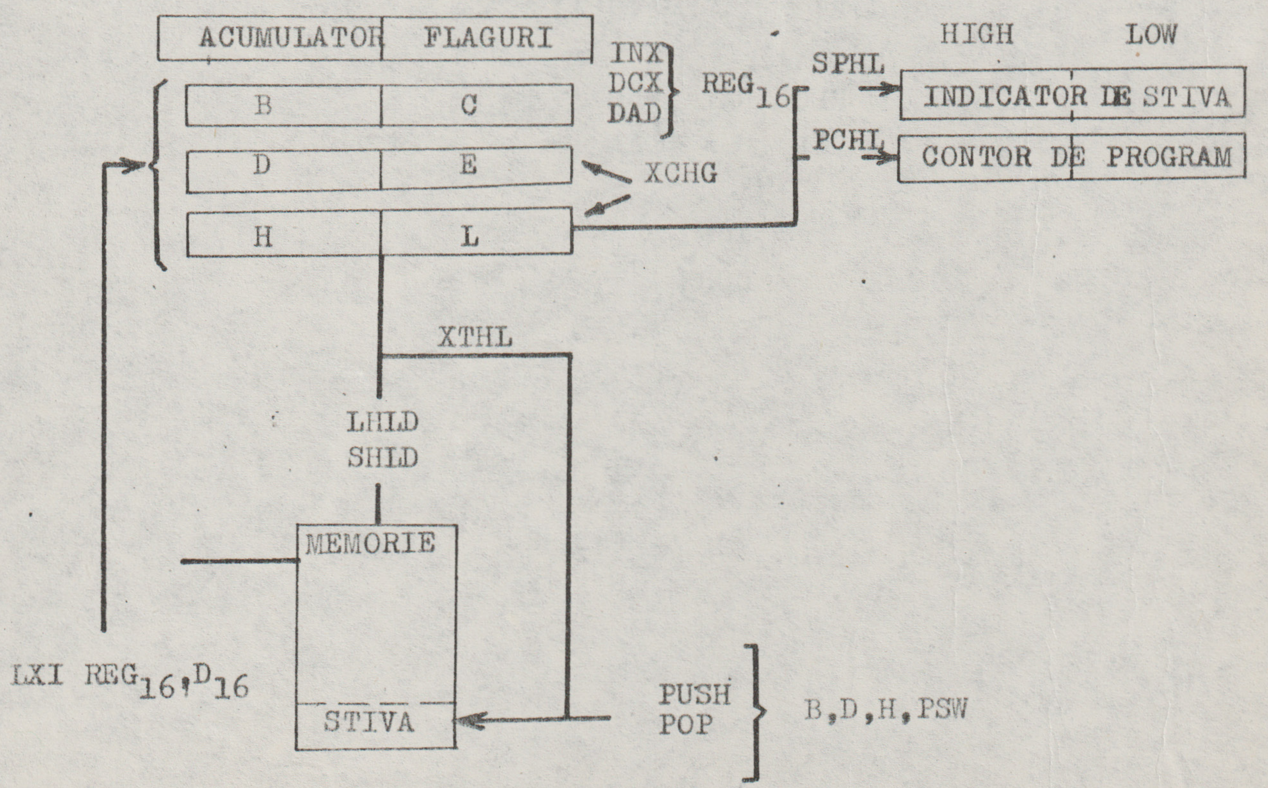
RAR CMA DAA

INR, DCR } $REGM_8$



Instrucţiuni pentru pereche de registre (cuvinte)

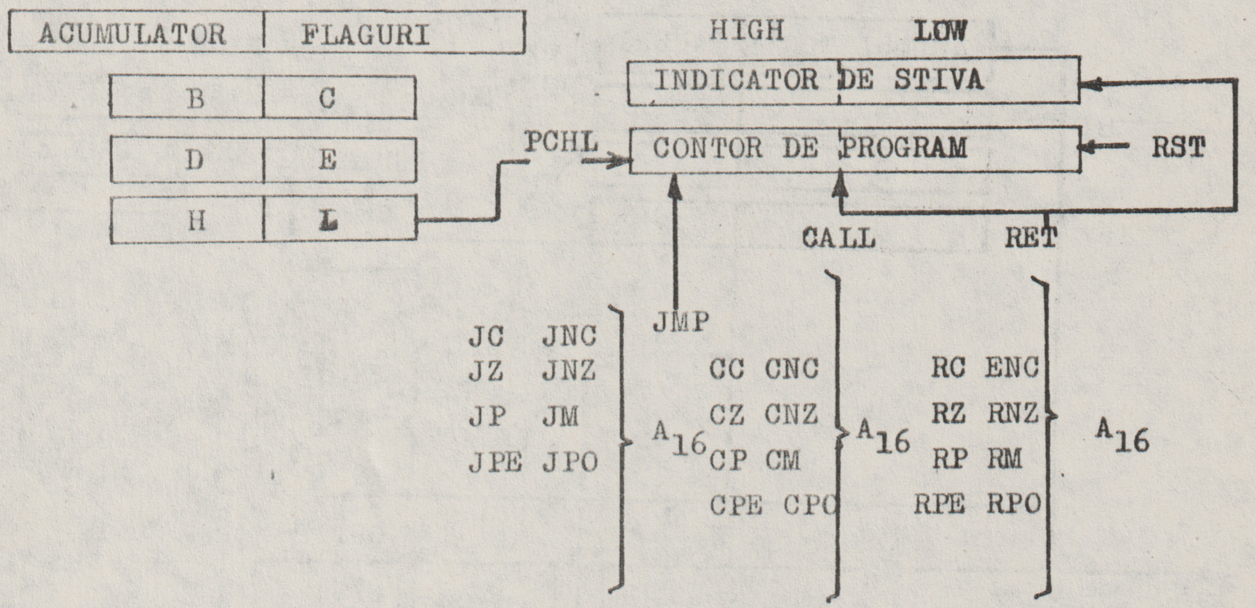
Următoarele instrucţiuni operează toate cu cuvinte de 16 biţi. Excepţie face numai instrucţiunea DAD (adună conţinutul registrului pereche B şi C sau D şi E la registrul H şi L). Nici una din aceste instrucţiuni nu afectează indicatorii de condiţie. Instrucţiunea DAD afectează numai indicatorul Carry.



Instrucţiuni de ramificare

Următoarele instrucţiuni pot modifica conţinutul contorului de program prin schimbarea fluxului normal de desfăşurare a programului. Instrucţiunile de salt afectează numai contorul de program.

Instrucţiunile CALL şi RETURN afectează contorul de program, indicatorul de stivă şi stiva.

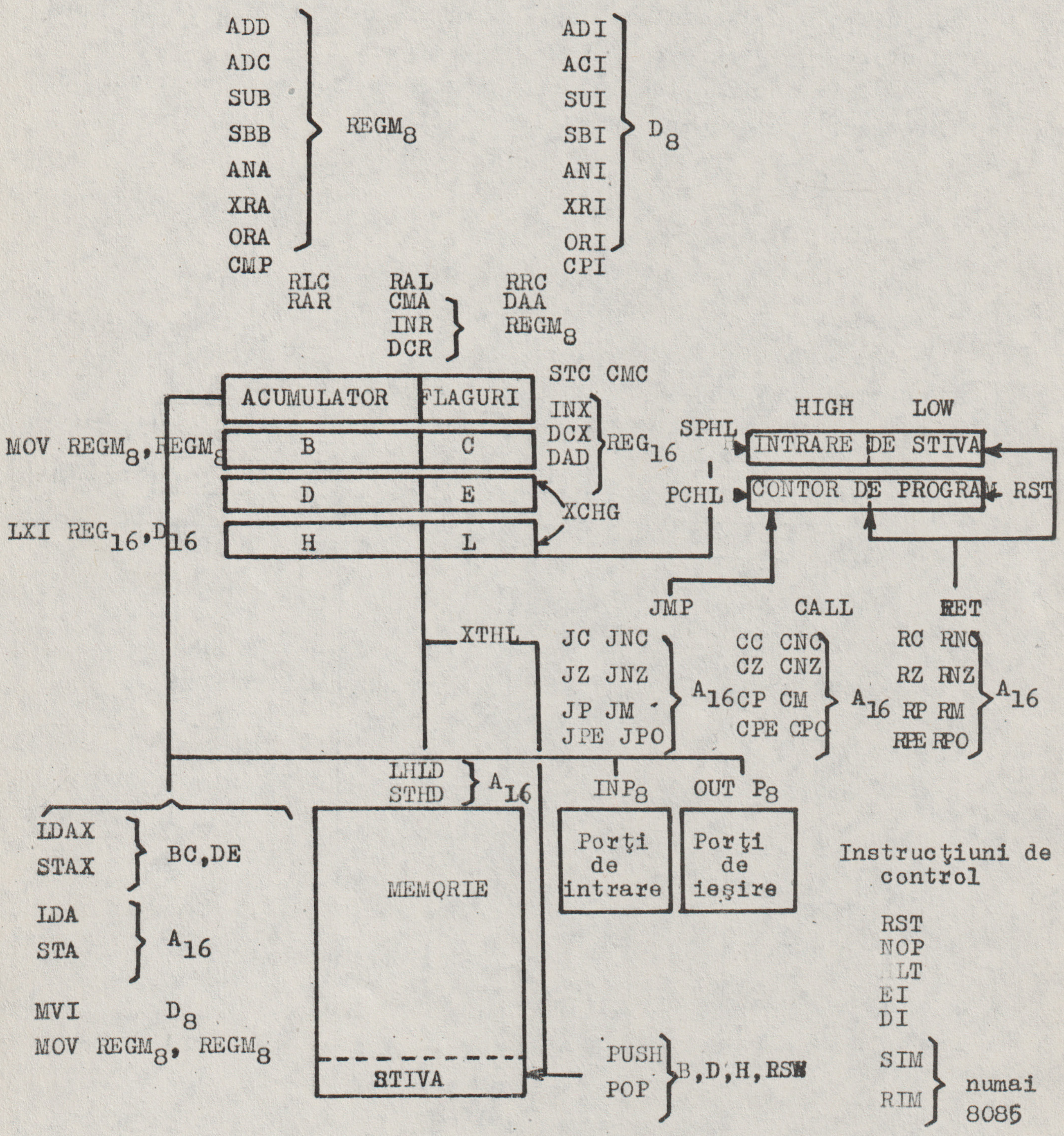


Instrucţiuni de comtrol

RST
NOP
HLT
EI
DI
SIM
RIM } 8085

<u>Ghidul setului de instrucţiuni</u>

In cele ce urmează este ilustrat rezultatul setului de instrucţiunie :



Codurile	Semnificaţia
$REGM_8$	- Operandul specifică unul din registrele de 8 biţi A,B,C,D,E.H,L sau memoria M (referitoare la memorie se face prin a-drese de 16 biţi conţinute în registrul H şi L). Instrucţiunea MOV care foloseşte doi operanzi poate specifica memoria nu - mai pentru unul din ei.
D_8	- Desemnează un operand imediat de 8 biţi.
A_{16}	- Desemnează adrese de 16 biţi.
P_8	- Desemnează numere de periferice de 8 biţi.
REG_{16}	- Desemnează un registru pereche de 16 biţi (B şi C, D şi E, H şi L sau SP)
D_{16}	- Desemnează un operand imediat de 16 biţi.

Diferenţele microprocesorului 8085

Diferenţele între microprocesorul 8080 şi 8085 sînt mai importante pentru proiectantul de sistem (Hardware) decît pentru programator (software). Cu excepţia a numai două instrucţiuni suplimentare, setul de instrucţiuni 8085 este identic şi total compatibil cu setul de instrucţiuni 8080.

In cele ce urmează sînt date facilităţile sistemului 8085

- o alimentare unică cu 5 V ;
- viteza de execuţie cu aproximativ 50 % mai rapidă faţă de 8080 ;
- încorporarea în procesor a generatorului de tact 8224, şi a controlorului de sistem şi magistrală 8228 ;
- întreruperi TRAP speciale (Non-maskable) pentru cazuti-le de defecţiuni în electroalimentare ;
- trei alte întreruperi separate (maskable) care generea - ză instrucţiuni interne RST ;
- linii de intrare/ieşire pentru transferul de date serie.

In cuprinsul descrierii, ori de cîte ori au existat diferenţe între sistemul 8080 şi 8085, acestea au fost evidenţiate

separat. Astfel sînt explicate instrucţiunile SIM şi RIM care sînt folosite de 8085 şi rutina de întreruperi.

2. CONCEPTELE LIMBAJULUI DE ASAMBLARE

Introducere

După cum limba engleză are regulile sale de gramatică, tot aşa limbajul de asamblare are reguli precise de codare.

Asamblorul recunoaşte trei tipuri de rînduri sursă: instrucţiunile, directivele şi controalele. In acest manual sînt descrise instrucţiunile şi directivele. Controalele sînt descrise în manualul de operare pentru această variantă de asamblare.

In acest capitol sînt descrise în general reguli pentru codarea rîndurilor sursă. Instrucţiunile specifice (a se vedea capitolul 3) şi directivele (a se vedea capitolul 4 şi 5) pot avea reguli specifice de codare. Astfel codarea instrucţiunilor şi directivelor trebuie să fie conforme cu regulile generale din acest capitol.

Formatul unei rînd sursă

Instrucţiunile limbajului de asamblare şi directivele de asamblare pot conţine pînă la patru cîmpuri după cum urmează :

Eticheta : Nume	CODUL DE OPERARE (OPCODE)	OPERANDUL	;COMENTARIUL

Cîmpurile pot fi separate printr-un număr de blancuri doar separarea trebuie să conţină cel puţin un delimitator . Fiecare instrucţiune şi directivă trebuie să fie cuprinsă într-un singur rînd care să se termine cu comanda de întoarcere a carului şi de schimbare a rîndului.

Continuarea rîndului nu este posibilă dar pot exista rînduri care să conţină în întregime comentarii.

Setul de caractere

Următoarele caractere sînt acceptate la exprimarea în limbaj de asamblare a sursei :

- literele alfabetului A la Z; fiind permise atît literele mari cît şi literele mici. Intern, asamblorul tratează toate literele ca litere mari, dar caracterele sînt imprimate în listingul de asamblare, exact cum au fost date la intrare;
- cifra de la 0 la 9 ;
- următoarele caractere speciale :

CARACTERUL	SEMNIFICATIA	CARACTERUL	SEMNIFICATIA
+	semn 1 plus	=	semnul egal
-	semnul minus		semnul mai mic
	asteriscul		semnul mai mare
/	bara de fracţie	%	procentul
,	virgula	!	semn exclamare
(	paranteza ded - chisă	BLANK	blanc sau spaţiu
)	paranteza închisă	;	punctul şi vir - gula
'	ghilimele	.	punct
&	şi	CR	întoarcere car
:	două puncte	FF	avans rînd
$	semnul dolar	HT	tabulare orizontală
@	semnul comercial "la"		
?	semn întrebare		

- în plus orice caracter ASCII poate apare într-un şir închis între ghilimele sau într-un comentariu.

Delimitări

Anumite caractere au un înţeles special pentru asamblor avînd funcţiuni de delimitare. Delimitatoarele definesc sfîrşitul unei declaraţii sursă, cîmpul, sau componente ale cîmpului.

In lista următoare sînt definite delimitatoarele recunoscute de asamblor. De notat că, unele delimitatoare referitoare la macro-instrucţiuni sînt explicate mai pe larg în capitolul 5.

CARACTERUL	SEMNIFICATIA	UTILIZAREA
blank	- Interval de separare între litere şi cuvinte	- Separator de cîmp sau simbol de terminare ;
,	- Virgula	- Separă operanzii în cîmpul operanzilor , include parametrii macro ;
'...'	- O pereche de ghilimele	- Delimitează un şir de caractere
(...)	- O pereche de paranteze	- Delimitează o expresie ;
CR	- Intoarcere car (carrioge return)	- Semn de terminare;
HT	- Tabulare orizontală (horizontal tab)	- Separator de cîmp sau simbol de terminare ;
;	- Punct şi virgulă	- Delimitator pentru cîmpul comentariului;
:	- Două puncte	- Delimitator pentru simbolurile folosite ca etichete;
&	- Si	- Delimitează textul prototip macro sau parametrii formali de înlănţuire;
<...>	- Pereche de paranteze unghiulare	- Delimitează textul cu parametru macro care conţine virgule sau blancuri; de asemenea se foloseşte pentru a delimita o listă cu parametri ;
%	- Semnul procent	- Delimitează,un parametru macro dacă este evaluată o substituire anterioară;
!	- Semnul exclamare	- Poate fi folosit pentru a trece un caracter ca parametru unui macro;
;;	- Pereche de punct şi virgulă	- Delimitează pentru comentarii în macro,definiţii cînd comentariul va fi suprimat cînd macro este extins

Cîmpul etichetei

Eticheta este întodeauna opţională. In cadrul instrucţiunii eticheta reprezintă o denumire simbolică a cărei valoare este locaţia unde instrucţiunea e asamblată.

O etichetă poate conţine de la unul pînă la şase caractere alfanumerice dar primul caracter trebuie să fie un caracter alfabetic ori unul din caracterele speciale "?" sau "@". Denumirea etichetei trebuie să fie terminată cu caracterul : (două puncte).

Simbolul folosit ca etichetă poate fi definit numai o singură dată în program.

Caracterele alfanumerice ce pot fi folosite într-o etichetă cuprind literele alfabetului, caracterul "semn de întrebare" şi cifrele zecimale 0 la 9.

O denumire de etichetă este necesară pentru directivele SET, EQU şi MARCO. Denumirile urmează aceleaşi reguli de codare ca etichetele cu deosebirea că denumirea trebuie să fie terminată cu caracterul "blank" în loc de caracterul două puncte.

Cîmpul etichetei numelui trebuie să fie egal pentru directivele LOCAL şi ENDM.

Cîmpul codului de operare

Acest cîmp conţine codul de operare prescurtat al instrucţiunilor pentru 8080/8085 sau directivele de asamblare ce trebuie îndeplinite.

Cîmpul operandului

Cîmpul operandului stabileşte datele ce trebuie să fie operate conform codului de operare ce a fost specificat în cîmpul anterior. Unele din instrucţiuni nu necesită operanzi;altele au unul sau doi operanzi.

In generalm cînd sînt necesari doi operanzi (de ex : transferul de date în operaţiunile aritmetice), primul operand identifică destinaţia (sau obiectivul) rezultatul operaţiunii şi cel de al doilea operand specifică datele sursă.

Exemple :

```
MOV A,C       ; Mută conţinutul registrului C în acumulator
MVI A, 'B'    ; Mută B în acumulator
```

Cîmpul comentariului

Cîmpul comentariului, care este opţional, poate conţine orice informaţie considerată ca necesară pentru adnotările din program. Singura condiţie ce se cere pentru acest cîmp este ca notaţia să fie precedată de caracterul; (punct şi virgulă).Din cauză că acest caracter este un delimitator nu trebuie să se separe comentariul de cîmpul anterior cu unul sau mai multe caractere "spaţiu". Oricum, spaţiile pot fi folosite, în cadrul comentariului, pentru a ameliora claritatea acestuia.

Deşi comentariul este opţional, se recomandă folosirea lui deoarece uşurează depanarea programului şi menţinerea unui program bine documentat.

Codarea informaţiilor din cîmpul operand

Există patru tipuri de informaţii (de la"a" la "d" în lista următoare) care pot fi cerute ca articole în cîmpul operand; informaţii ce pot fi exprimate în nouă moduri diferite ce vor fi descrise în cele ce urmează :

Informaţia cîmpului operand

Informaţia cerută	Modul de exprimare
a) Registru	1) Date în hexazecimal
b) Registru pur	2) Date în zecimal
c) Date directe	3) Date în octal
d) Adresa de 16 biţi	4) Date în binar
	5) Adresa contor ($)
	6) Constantă ASCII
	7) Etichetele asignate valorilor
	8) Etichetele instrucţiunilor sau datelor
	9) Expresii

1) Date în hexazecimal

Fiecare număr hexazecimal trebuie să înceapă cu o cifră numerică (0 la 9) şi trebuie să fie urmată de litera H.

Eticheta	Codul de operare	Operandul	Comentariul
HERE:	MVI	C,0BAH	;Incarcă registrul cu valoarea AB în hexazecimal

2. Date în zecimal

Fiecare număr poate fi identificat prin litera D imediat după ultima sa cifră sau aceasta poate lipsi. Orice număr la care nu este specificată exprimarea făcută în hexazecimal, octal sau binar este considerat zecimal. Astfel se dau următoarele exemple :

Eticheta	Codul de operare	Operandul	Comentariul
ABC:	MVI	E,15	;Incarcă E cu 15 zecimal
	MVI	E,15 D	

3. Date în octal

Fiecare număr octal trebuoe să fie urmat de litera O sau litera Q.

Eticheta	Codul de operare	Operandul	Comentariul
LABEL:	MVI	A,72 Q	;Incarcă numărul octal 72 în acumulator

4. Date în binar

Fiecare număr binar trebuie să fie urmat de litera B

Eticheta	Codul de operare	Operandul	Comentariul
NOW:	MVI	D,11110110 B	;Incarcă numărul OF6H în registrul D;

5. Contorul de adresă

Caracterul $ se referă la adresa curentă a contorului. Contorul de adresă conţine adresa unde va fi asamblată instrucţiunea curentă sau datele.

Ericheta	Codul de operare	Operandul	Comentariul
GO:	JMP	$+6	;Salt cu 6 b peste adresa ;primului bait a acestei ins- ;trucţiuni

6. Constante ASCII

Unul sau mai multe caractere ASCII cuprinse între ghilimele definesc o "constantă ASCII".

Eticheta	Codul de operare	Operandul	Comentariul
	MVI	E,/*/	;Incarcă registrul E cu caracterul semnului * reprezentat ;prin 8 biţi în ASCII;
DATE	DB	'TODAY"S"DATE'	

7. Etichetele valorilor asigurate

Directivele SET şi EQU pot asigna valori la etichete. In exemplu următor, eticheti VALUE ia fost asignată valoarea 9FH. Următoarele două,exemple sînt echivalente:

Eticheta	Codul de operare	Operandul	Comentariul
A1:	MVI	D,9FH	
A2:	MVI	D,VALUE	

8. Etichetele instrucţiunilor sau datelor

Eticheta asignată la instrucţiune sau definire de date are valoarea de adresă a primului bait al instrucţiunii sau da - telor. Astfel de instrucţiuni în program pot referi o adresă printr-o etichetă simbolică.

Eticheta	Codul de operare	Operandul	Comentariul
HERE:	JMP	THERE	;Salt la ins - ;trucţiunea de ;la THERE
THERE:	MVI	D,9FH	;Incarcă numă- ;rul 9FH în re- ;gistrul D;

9. Expresii

Toate tipurile de operanzi discutate pînă aici pot fi combinate prin operatori pentru a forma o expresie. De fapt,ex-

emplul dat pentru adresa contor ($+6) reprezintă o expresie. Deoarece codarea expresiilor este foarte variată, în viitoarele discuţii din acest capitol va fi reluată.

Instrucţiuni operanzi

Un tip de operand a fost intenţionat omis de pe lista informaţiilor din cîmpul operand. Instrucţiunile cuprinse între paranteze pot apare în cîmpul operanzilor. Operandul în acest caz, are valoarea baitului celui mai din stînga a instrucţiunii asamvlate.

Eticheta	Codul de operare	Operandul
INS:	DB	(ADD C)

Declaraţia de mai înainte defineşte baitul cu valoarea 81 H (codul obiect pentru instrucţiunea ADD C). Astfel de codare e tipic folosită unde programul obiect modifică durata execuţiei programului, tehnică puternic descurajată.

Operand tip - registru

Numai instrucţiunile care conţin registre în cîmpul operandului pot avea operand tip-registru.

Expresiile conţinînd operanzi tip-registru sînt evidenţiate ca erori. Astfel instrucţiunea

JMP A

este semnalizată ca folosire ilegală a registrului. Nu numai directivele asamblor ce pot conţine operand tip-registru sînt EQU, SET şi parametrii din apelurile macro.

Registrele pot fi asigurate cu alte nume numai prin EQU sau SET.

Reprezentarea datelor în complement de 2

Orice bait de 8 biţi, conţine una din cele 256 combinaţii posibile de 0 şi 1. Anumite combinaţii pot fi interpretate în mai multe feluri. De exemplu, codul 1FH poate fi interpretat ca o instrucţiune (RAR, rotire acumulator la dreapta cu transport), ca valoare hexazecimală 1F, ca valoare zecimală 31 sau ca valoare binară 00011111.

Instrucţiunile aritmetice presupun că baiţii de date

asupra cărora operează sînt în format "complement de 2 ".

Pentru înţelegere, mai întîi se vor examina două exemple în aritmetica zecimală :

$$\begin{array}{r} 35 \\ -\ 12 \\ \hline 23 \end{array} \qquad \begin{array}{r} 35 \\ +\ 88 \\ \hline 1\ 23 \end{array}$$

De notat că re ultatul acestor două exemple este egal dacă se ignorează transportul cifrei 1 spre poziţia de ordin superior a celui de al doilea exemplu.

Cel de al doilea exemplu ilustrează scăderea din primul exemplu, realizată prin adunarea complementului de zece al scăzătorului 12. Complementul lui zece al unui număr zecimal se obţine scăzînd, în primul rînd, fiecare cifră a scăzătorului din cifra 9, obţinîndu-se complementul de nouă,apoi adunîndu-se cifra 1 la rezultat se obţine complementul de zece. Astfel , 99-12= 87 ; 87 + 1 = 88, 88 = complementul de 10 al lui 12.

Posibilitatea efectuării scăderii sub formă de adunare reprezintă un mare avantaj în calculatoare, deoarece reduce numărul circuitelor electronice necesare. De asemenea, operaţiunile în calculator sînt binare, ceea ce simplifică mai mult problemele.

Procesorul realizează complementul de doi al unei valori binare, simplu prin inversarea fiecărui biţ şi apoi prin adunarea cifrei 1 la rezultat.

După ce complementul este format, orice transport rezultat în bitul de ordine superioară este ignorat.

Apoi scăderea este efectuată după cum se arată în continuare :

$$\begin{array}{l} 35 = 00100011 \\ -\ 12 = 00001100 = 11110011 \\ \qquad\qquad\qquad\quad +\qquad\quad 1 \\ \hline 23 \qquad\qquad\qquad 11110100 \end{array} \qquad \begin{array}{r} 00100011 \\ +\ 11110100 \\ \hline 1\ 00010111 = 23 \end{array}$$

Astfel, prin ignorarea baitului de transport din poziţia de ordin superior, scăderea este efectuată sub forma unei adunări. Cu toate acestea, dacă operaţia a fost făcută de microprocesorul 8080 sau 8085 indicatorul transportului va fi făcut 0 la termina-

rea scăderii. Acesta din cauză că procesorul complementează indicatorul de transport la sfîrşitul scăderii, încît să poată fi folosit ca indicator de "împrumut" în scăderile multibait. In exemplu arătat nu este necesară un împrumut aşa că indicatorul transportului a fost setat OFF. Spre deosebire, indicatorul transportului este setat ON dacă se scade 35 din 12 :

```
   12 = 00001100                  00001100
 - 35 = 00100011 = 11011100     + 11011101
                 +        1
                 ----------     ----------
                   11011101       11101001 = 233 sau - 105
```

In acest caz, lipsa transportului indica necesitatea împrumutului de la următorul bait de ordine superioară dacă există. Procesorul a setat ON indicatorul transportului. Trebuie reţinut de asemenea că rezultatul este stocat sub forma complementată.

Dacă se doreşte ca acest rezultat să fie interpretat în valoare zecimală se poate pune din nou sub,formă de complement de doi :

```
11101001 = 00010110
         +        1
         ----------
           00010111 = 23
```

Numărul reprezentat în complementul în doi poate avea semn. Cînd baitul este interpretat ca număr complement de doi cu semn, bitul de ordin superior reprezintă semnul. Un zero în acest bait indică un număr pozitiv, iar un unu număr negativ.

Următorii şapte biţi de ordin inferior reprezintă mărimea numărului.Astfel, 01111111 este egal cu + 127 şi 11111111 este egal cu - 127.

La începutul prezentării acestui capitol al complementului de doi s-a arătat că un bait de 8 biţi poate conţine una din cele 256 combinaţii posibile de 0 şi 1. Trebuie de asemenea precizat că interpretarea particularităţii datelor cade în sarcina programării.

Un exemplu poate fi considerat instrucţiunea COMPARE. Compararea logică consideră că numai valorile primului bait să

fie comparate. In aceste condiţii există cazuri de nedetermina-re ce trebuie să fie eliminate printr-un program corespunzător.

Simboluri şi tabele de simboluri

Adresare simbolică

Dacă nu se cunoaşte încă noţiunea de programare simboli_că, următoarea analogie va ajuta să se facă o distincţie clară între adresa simbolică şi adresa absolută.

Locaţiile din memorie program pot fi comparate cu grupul de boxe de la căsuţa poştală a oficiului poştal.

Se presupune boxa 500 închiriată pentru două luni de persoana X. Aceasta poate cere pentru a primi corespondenţa ce i-a fost adresată : daţi-mi corespondenţa din boxa 500, sau daţi-mi corespondenţa pentru X. Dacă mai tîrziu, o altă persoană Y în - chiriază boxa 500, acesta va cere: daţi-mi corespondenţa din boca 500 sau corespondenţa pentru X.

Conţinutul căsuţei poştale poate fi accesat prin o adresă absolută, fixă (500) sau una simbolică, variabilă.

Oficiantul de la oficiul poştal corelează denumirile simbolice şi valorile lor absolute în registrul său.

Asamblorul îndeplineşte acelaşi funcţiuni, păstrează în tabela de simboluri evidenţa legăturilor dintre denumirile simbolice şi valorile absolute. De notat că nu se asignează valori numerice la adresele simbolice. In timpul procesului de asamblare, numărătorul de locaţii dă referirile de asamblare pentru ceea ce face contorul de program pentru microcomputer. El anunţă asamblorul unde următoarea instrucţiune sau operand trebuie să fie plasată în memorie).

Caracteristicile simbolurilor

Un simbol poate conţine de la unu la şase caractere alfabetice (A-Z), sau numerice (0-9), (primul caracter fiind alfabetic), sau caracterul special '? ''ori @' .

Semnul dolarului poate fi folosit ca simbol pentru a nota valoarea curentă în numărătorul de locaţii.

De exemplu, comanda

JMP $ + 6

forţează un salt la instrucţiunea ce se găseşte cu 6 locaţii

de memorie mai sus de instrucţiunea JMP.

Simbolurile de forma '?? nnn''sînt generate de asamblor ca denumiri unice de simboluri locale pentru MACRO.

Asamblorul verifică dacă simbolurile au următoarele atribute; rezervate sau definite de utilizator, globale sau limitate, permanente sau redefinite şi absolute sau relocabile.

Simboluri rezervate, definite de utilizator şi generate de asamblor

Simbolurile rezervate sînt toate acelea care au un înţe - les special pentru asamblor şi care nu pot fi definite de de utilizator.

Denumirile prescurtate pentru instrucţiunile maşină, şi pentru directivele de asamblare sînttoate simbolurile rezervate.

Următoarele simboluri de operanzi de instrucţiuni sînt de asemenea rezervate :

Simbolul	Semnificaţia
$	Referinţă la numărătorul de locaţie
A	Registrul acumulator
B	Registrul B sau perechea de registre B şi C
C	Registrul C
D	Registrul D sau perechea de registre D şi E
E	Registrul E
H	Registrul H sau perechea de registre H şi L
L	Registrul L
SP	Registrul indicator de stivă
PSW	Cuvîntul de stare a procésorului (A şi indicatorii)
M	Cod de referinţă la memorie, folosind adrese în H şi L
STACK	Cu caracter special de relocare
MEMORY	Cu caracter special de relocare

Nota : Simbolurile STACK şi MEMORY sînt discutate pe larg în capitolul 4

Simbolurile definite de utilizator sînt acele simboluri create pentru referirea la instrucţiuni şi adrese de date. Aceste simboluri sînt definite cînd apar în cîmpul etichetă al instrucţiunii sau al directivelor EQU, SET ori MACRO în cîmpul nume (vezi capitolele 4 şi 5).

Simbolurile globale şi limitate

Majoritatea simbolurilor sînt globale. Acestea înseamnă au aceeaşi semnificaţie peste tot în program. Să presupunem, spre exemplu, că se asignează unei rutine denumirea simbolică RTN. Se poate astfel coda un salt sau un apel la RTN din orice punct al programului. Dacă se asignează denumirea simbolică RTN la o a doua rutină va rezulta o eroare, deoarece vor exista mai multe definiţii pentru acelaşi nume.

Anumite simboluri au semnificaţie numai în cadrul definiţiilor macro sau în codul apelurilor către macro; aceste simboluri sînt "locale" în macro.

Macro necesită simboluri locale, deoarece acelaşi macro poate fi folosit de mai multe ori în program. La fiecare folosire a acestui macro (cu excepţia primei) se vor produce erori,dacă numele simbolic era global.

La capitolul 5 sînt prezentate informaţii suplimentare despre macro.

Simbolurile permanente şi redefinibile

Majoritatea sombolurilor sînt prezentate deoarece valoarea lor nu poate fi schimbată în timpul operaţiunilor de asamblare. Numai simbolurile pentru directivele asambler SET şi MACRO sînt redefinibile.

Simbolurile absolute şi relocabile

Un important atribut al simbolurilor în asamblare este relocabilitatea. Programele relocabile sînt asamblate în memorie de la locaţia zero. Aceste programe sînt relocate ulterior în alte set de locaţii din memorie.

Simbolurile care îşi schimbă adresa în timpul relocării sînt simboluri relocabile. Simbolurile care nu îşi schimbă adresa în timpul relocării sînt simboluri absolute.

Aceste distincii încep să devină importante cînd simbolurile sînt folosite în expresii, ceea ce se va explica mai tîrziu.

Evaluarea expresiei în timpul asamblării

O expresie este o combinaţie de numere, simboluri de operatori. Fiecare element al expresiei reprezintă un termen.

Expresiile, ca şi simbolurile, pot fi absolute sau relocabile.

Operatorii

Asamblorul cuprinde cinci grupe de operatori care permit următoarele operaţiuni :

- operaţiuni aritmetice
- operaţiuni de deplasare
- operaţiuni logice
- operaţiuni de comparare
- operaţiuni de izolare a baitului.

Este important de ştiut că acestea sînt toate operaţiunile de asamblare. De îndată ce asamblorul a evaluat o expresie, ea începe să facă în permanenţă parte din program. Să presupunem, de exemplu, că programul dumneavoastră defineşte o listă cu zece constante începînd de la eticheta LIST ; următoarea instrucţiune încarcă adresa articolului al 7-lea din listă în registrele H şi L :

LXI H, LIST + 6

De notat că adresa primului articol din listă este LIST, a celui de al doilea articol LIST + 1 ş.a.m.d.

Operatori aritmetici

Operatorii aritmetici sînt următorii:

Operatorul	Semnificaţia
+	- Adunarea unară sau binară
-	- Scăderea unară sau binară
*	- Inmulţire (multiplicare)
/	- Impărţire (divizare). De reţinut că resturile sînt eliminate (7/2 = 3) iar împărţirea cu zero provoacă o eroare.

Operatorul	Semnificaţia
MOD	- Modulo. Rezultatul este restul obţinut de la operaţiunea de împărţire (7 MOD 3 = 1).

Exemple :

Următoarele expresii generează succesiunea de biţi pentru caracterul A din codul ASCII :

5 + 30*2
(25/5) + 30*2
5+(-30* -2)

De notat că operatorul MOD trebuie să fie separat de operandul său prin spaţii.

NUMBR MOD 8

Presupunînd că NUMBR are valoarea 25, expresia anterioară e evaluată la valoarea 1.

Operatori de deplasare

Operatorii de deplasare sînt următorii :

Operatorul	Semnificaţia
Y SHP x	Deplasează operatorul "Y" către dreapta cu "x" poziţii de biţi ;
Y SHL x	Deplasează operatorul "Y" către stînga cu "x" poziţii de biţi

Operatorii de deplasare nu trebuie să rotească biţii deplasaţi afară din bait. Poziţiile părăsite de biţi, în operaţiunea de deplasare sînt umplute cu zero.

De notat că operatorul de deplasare trebuie să fie separat de operanzii săi prin spaţii.

Exemple :

Se presupune că NUMBR are valoarea 01010101. Efectele operaţiilor de deplasare :

```
NUMBR SHR 2      ;Deplasează operandul 01010101
                 ;către dreapta cu 2 poziţii de biţi

NUMBR SHL 1      ;Deplasează operandul 01010101 către
                 ;stînga cu un bit.
```

sînt următoarele :

```
0101 0101        0001 0101

0101 0101        1010 1010
```

De notat că pentru valorile ne-negative, o deplasare cu un bit către stînga, are ca efect o multiplicare cu doi a valorii respective, iar o deplasare cu un bit către dreapta are ca efect o împărţire a valorii cu doi.

Operatorii logici

Operatorii logici sînt următorii :

Operatorul	Semnificaţia
NOT	Complementul logic
AND	SI logic (=1 dacă ambii biţi sînt 1)
OR	SAU logic (=1 dacă cel puţin un bit este 1)
XOR	SAU exclusiv (=1 dacă biţii sînt diferiţi)

Operatorii logici acţionează numai asupra bitului cel mai puţin semnificativ al rezultatului operaţiei. De asemenea, aceşti operatori sînt folosiţi în directivele IF condiţionate. Aceste directive sînt explicate pe larg în capitolul 4.

Exemple :

Următoarea directivă IF testează bitul cel mai puţin semnificativ al expresiei evoluate. Codul limbajului de asamblare al celor ce urmează după IF este asamblat numai dacă condiţia este adevărată (TRUE).

```
IF FLDI AND FLD2 AND FLS 3
```

Operatori de comparare

Operatorii de comparare sînt următorii :

Operatorul	Semnificaţia
EQ	Egalitate
NE	Inegalitate
LT	Mai mic
LE	Mai mic sau egal
GT	Mai mare
GE	Mai mare sau egal
NUL	Operator special folosit pentru a testa anularea (eliminarea) parametrilor macro.

Operatorii de comparare produc un rezultat da - nu. Astfel, dacă relaţia evaluată este adevărată, valoarea rezultatului este : toţi biţii unu. Dacă este falsă toţi biţii sînt zero.

Operaţiunile de relaţie sînt strict bazate pe mărimea valorilor biţilor comparaţi. Astfel, complementul de doi al unui număr negativ (care are un unu poziţia bitului de pondere superioară) este mai mare de cît complementul de doi al unui număr pozitiv (care are totdeauna un zero în poziţia bitului de pondere superioară).

Operatorul NUL are aplicaţii numai în MACRO.

NUL este descris în capitolul 5.

Operatorii de comparare sînt folosiţi în directivele IF condiţionate. Aceste directive sînt explicate pe larg în capitolul 4.

De notat că, operatorul de comparare trebuie să fie separat de operanzii săi prin spaţii.

Exemplu :

Următoarele directive IF tratează egalitatea dintre valorile lui FLD 1 şi FLD 2. Dacă rezultatul comparaţiei este TRUE codul limbajului de asamblare a celor ce urmează după directiva IF este asamblat. In caz contrar, codul este sărit.

```
IF FLD1 EQ FLD2
      :
```

Operatorii de izolare a baitului

Operatorii de izolare a baitului sînt următorii :

Operatorul	Semnificaţia
HIGH	Izolează cei 8 biţi de pondere superioară din o valoare de 16 biţi ;
LOW	Izolează cei 8 biţi de pondere inferioară din o valoare de 16 biţi.

Asamblorul tratează expresiile ca adrese de 16 biţi. In unele cazuri se doreşte să se folosească numai o parte a adresei sau să se genereze o valoare de 8 biţi.

Aceasta este funcţia operatorilor HIG sau LOW.

Exemple :

Să presupunem că ADRS este o adresă manipulată în asamblor pentru formarea de tabele sau liste de articole care trebuie să fie toate sub adresa 255 din memorie. Următoarea directivă IF determină, dacă baitul de ordine superioară al ADRS este zero, inidînd astfel că adresa este mai mică decît 256 :

IF HIGH ADRS EQ O

.
.
.

Rangul permis al valorilor

Intern, asamblorul tratează fiecare termen al unei expresii ca pe doi baiţi, adică o valoare de 16 biţi. Astfel,rangul maxim al valorilor este OH pînă la OFFFFH.

Toate operaţiunile aritmetice sînt făcute folosindu-se complementul faţă de doi aritmetic fără semn.

Asamblorul nu realizează o detecţie a depăşirii valorilor de doi baiţi, aşa încît aceste valori sînt evaluate modulo 64 K.

Anumite instrucţiuni necesită ca operatorii lor să aibă valoarea a 8 biţi. Expresiile pentru aceste instrucţiuni trebuie să conţină valori cuprinse între - 256 şi + 255.

Asamblorul, în cazul cînd o expresie pentru una din aceste instrucţiuni are o valoare în afara domeniului - 256 la + 255, generează o eroare.

Prioritatea operatorilor

Expresiile sînt evaluate de la stînga la dreapta. Operatorii cu prioritate superioară sînt evaluaţi înaintea altor operatori care îi preced sau îi urmează.

Cînd doi operatori au prioritate egală, primul este evaluat cel mai din stînga.

Parantezele pot fi folosite pentru a sări regulile normale de prioritate. Partea unei expresii care este inclusă între paranteze este evaluată întîi. Dacă parantezele sînt cuprinse în cadrul altor paranteze, cele din interior sînt evaluate mai întîi.

15/3 P 18/9 = 5 + 2 = 7

15/(3+18/9) = 15(3 + 2) = 15/5 =,3

In lista următoare sînt descrise clasele operatorilor în ordinea priorităţilor :

- Expresiile din paranteze
- NUL
- HIGH, LOW
- Inmulţirea / împărţirea : *,/, MOD,SHL, SHR
- Adunarea / scăderea : +, - (unar şi binar)
- Operatorii relaţionali : EQ, LT, LE, GT, GE, NE
- NOT logic
- AND logic
- OR, XOR logic

Operatorii relaţionali, logici şi HIGH/LOW trebuie să fie separaţi de operanzii lor prin cel puţin un blanc.

Atenţie

Dacă NOT este imediat precedat de un alt operator, ca de exemplu:

Y EQU 1 + NOT X

va rezulta o eroare.Pentru asamblarea corectă a expresiei, expresia se pune între paranteze pentru a forţa ca NOT să fie evaluat întîi, astfel:

Y EQU 1 + (NOT X)

Expresii relocabile

Stabilirea relocabilităţii unei expresii necesită să fie înţeleasă relocabilitatea fiecărui termen folosit în expresie.

Acest lucru este mai uşor dacă numărul operatorilor este mic, dar în primul rînd este necesar să se cunoască dacă un simbol este absolut sau relocabil.

Simbolurile absolute pot fi definite în două moduri :

- Un simbol care apare în cîmpul etichetei cînd directiva ASEG este în desfăşurare este un simbol absolut.
- Un simbol definit ca echivalent cu o expresie absolută folosind directiva SET sau EQU este un simbol absolut.

Simbolurile relocabile pot fi definite în mai multe moduri :

- Un simbol care apare în cîmpul etichetei cînd directiva DSEG sau CSEG este în desfăşurare este un simbol relocabil.
- Un simbol definit ca echivalent cu o expresie relocatabilă folosind directiva SET sau EQU este relocabil.
- Simbolurile speciale de asemblere STACK şi MEMDRY sînt relocabile.
- Simbolurile externe sînt considerate relocabile.
- Referirea la contorul de locaţii (specificat de caracterul $) este relocabilă dacă o directivă CSEG sau DSEG este în desfăşurare.

Expresiile arătateîn lista următoare sînt numai expresii care produc rezultate relocabile, Presupune, că ABS este un simbol absolut şi RELOC este un simbol relocabil.

Se reţine că numerele sînt termeni absoluţi. Astfel, expresia RELOC - 100 este admisă (legală), dar expresia 100-RELOC nu este.

Cînd două simboluri relocabile au fost definite cu acelaşi tip de relocabilitate, acestea pot fi conţinute în anumite expresii care produc un rezultat absolut. Simbolurile au acelaşi tip de relocabilitate cînd ambele sînt legate de contorul de locaţii CSEG, ambele sînt legate de contorul de locaţie DSEG, ambele sînt legate de MEMORY, sau ambele sînt legate de STACK.

Următoarele expresii sînt posibile şi produc rezultate absolute :

RELOC 1 - RELOC 2

RELOC1 { EQ / LT / LE / GT / GE / NE } RELOC2

Simbolurile relocabile nu pot fi cuprinse în expresii cu oricare alţi operatori.

In lista următoare sînt arătate toate combinaţiile posibile ale operatorilor cu termeni absoluţi şi relocabili.

Caracterul A din tabelă indică o adresă absolută : R indică o adresă relocabilă; I indică o combinaţie ilegală (neadmisă). De notat că ultimii cinci operatori din listă pot apare numai cu un singur termen.

Operatorul	X absolut Y absolut	X absolut Y relocabil	X relocabil Y absolut	X relocabil Y relocabil
X + Y	A	R	R	I
X - Y	A	I	R	A
X * Y	A	I	I	I
X/Y	A	I	I	I
X MOD Y	A	I	I	I
X SHL Y	A	I	I	I
X SHR X	A	I	I	I
X EQ Y	A	I	I	A
X LT Y	A	I	I	A
X LE Y	A	I	I	A
X GT Y	A	I	I	A
X GE Y	A	I	I	A
X NE Y	A	I	I	A
X AND Y	A	I	I	I
X OR Y	A	I	I	I
X XOR Y	A	I	I	I
NOT X	A	-	I	-

Operatorul	X absolut Y absolut	X Y absolut relocare	X Y relocabil absolut	X relocabil Y relocabil
HIGH X	A	-	R	-
LOW X	A	-	R	-
unitatea +X	A	-	R	-
unitatea -X	A	-	I	-

Relocabilitatea expresiilor cu simboluri externe

Sînt permise numai următoarele expresii ce conţin simboluri externe (EXTRN) :

- Simbol extern ± simbol absolut
- Simbol absolut + simbol extern
- HIGH (simbol extern)
- LOW (simbol extern)

Inlănţuirea definiţiilor simbolurilor

Macro asamblorul ISIS-II 8080/8085 este în esenţă un asamblor în doi paşi. Toate intrările în tabela de simboluri trebuie să fie rezolvabile în doi paşi.

Prin urmare seria

```
X EQU Y
Y EQU 1
```

este ilegală, iar în seriile

```
X EQU Y
Y EQU Z
Z EQU 1
```

primul rînd este ilegal deoarece X nu poate fi rezolvat în doi paşi şi rămîne nedefinit,

CAPITOLUL 3

3. SETUL DE INSTRUCTIUNI

Acest capitol constituie un dicţionar al instrucţiunilor 8080 şi 8085. Descrierea instrucţiunilor este listată în ordine alfabetică pentru a se putea face referiri rapide.

Informaţii de timp

Descrierea instrucţiunilor din acest manual nu cuprinde date privind duratele de execuţie. Acasta din cauză că viteza de operare în procesorul folosit depinde de frecvenţa de tact utilizată în sistem.

"Starea" constituie unitatea de bază pentru măsurarea timpului de către procesor . Această unitate de timp "Stare" este cuprinsă între 48o nanosecunde (320 nanosecunde pentru 8085) şi 2 microsecunde, în funxţie de frecvenţa de tact. Dacă se cunoaşte lungimea unităţii de timp într-un anumit sistem, se poate determina timpul de execuţie al unei instrucţiuni înmulţindu-se această durată cu numărul de stări necesare instrucţiunii.

ACI Adunarea imediată cu transport

Instrucţiunea ACI adună conţinutul celui de al doilea bait al instrucţiunii şi bitul de transport la conţinutul acumulatorului şi stochează rezultatul în acumulator.

Codul de operare	Operandul
ACI	data

Operandul specifică datele respective ce vor fi adunate la acumulator cu excepţia, de sigur, a baitului de transport. Datele pot fi de forma unui număr, o constantă ASCII, eticheta unei valori d finite anterior sau o expresie.

Datele nu pot depăşi un bait (byte).

Asamblorul se caracterizează prin tratarea tuturor simbolurilor externe şi relocabile ca adrese de 16 biţi.

Cînd unul din aceste simboluri este conţinut în expresia operand a instrucţiunii imediate, ea trebuie să fie precedată de un operator HIGH ori LOW care să specifice care bait din adresă

trebuie să fie folosit pentru evaluarea expresiei.

Cînd nu este dat nici-un operator asamblorul consideră un operator LOW şi este dat un mesaj de eroare.

CEH =

1 1 0 0 1 1 1 0
Date

Cicluri : 2

Stări : 7

Adresare : imediată

Indicatori de condiţie afectaţi (flaguri afectate) : Z,S,P,CY,AC

Exemple :

Se presupune că acumulatorul conţine valoarea 14 H şi că bitul de transport are valoarea 1. Instrucţiunea ACI 66 are următorul efect :

Acumulatorul	=	14 H	00010100
Date imediate	=	42 H	01000010
Transport	=		1
			01010111 = 57 H

<u>ADC</u> <u>Adunare cu transport</u>

Instrucţiunea ADC adună un bait de date plus transportul la conţinutul acumulatorului, Rezultatul este stocat în acumulator. ADC actualizează apoi indicatorul CY pentru a indica transportul operaţiei efectuate.

Instrucţiunea ADC foloseşte bitul de transport pentru a face posibilă adunarea multiplă de baiţi (byte-ţi).

<u>Adună conţinutul unui registru la acumulator cu transport</u>

<u>Codul de operare</u>	<u>Operandul</u>
ADC	registru

Operandul trebuie să specifice unul din registrele A la E, H sau L. Această instrucţiune adună conţinutul registrului

specificat şi a bitului de transport la acumulator şi apoi stochează rezultatul în acumulator.

88 H ... 8 DH sau 8 FH = | 1 0 0 0 1 | S S S |

Cicluri : 1
Stări : 4
Adresare : registru
Flaguri afectate: Z,S,P,CY,AC

<u>Adună conţinutul memoriei la acumulator cu transport</u>

Codul de operare	Operandul
ADC	M

Această instrucţiune adună conţinutul locaţiei din memorie adresată de registrele H şi L plus bitul de transport la acumulator şi apoi stochează rezultatul în acumulator. M este un simbol de referinţă la registrele H şi L.

8 EH = | 1 0 0 0 1 1 1 0 |

Cicluri : 2
Stări : 7
Adresare : : registru indirect
Flaguri afectate : Z,S,P,CY,AC

Exemplu :

Se presupune că registrul C conţine 3 DH, acumulatorul conţine 42 H şi bitul CY de transport este zero.

Instrucţiunea ADC C realizează adunarea în felul următor :

3DH = 00111101
42H = 01000010
CY =, 0
01111111 = 7FH

Indicatorii de condiţii rezultaţi sînt :

Z = 0, S = 0, P = 0, CY = 0 AC = 0

Dacă bitul de transport CY are valoarea 1, instrucţiunea are următorul rezultat :

3DH = 00111101		Z = 0
42H = 01000010		S = 1
CY = 1		P = 0
10000000 = 80 H		CY = 0
		AC = 1

ADD

Instrucţiunea ADD adună un bait de date la conţinutul acumulatorului. Rezultatul este stocat în acumulator. De notat că instrucţiunea ADD nu adună şi bitul de transport (CY), dar setează flagul pentru a indica rezultatul operaţiei.

Adună registrul la registru :

Codul de operare	Operandul
ADD	reg

Operandul trebuie să specifice unul din regisrele A la E, H sau L.

Instrucţiunea adună conţinutul registrului specificat la conţinutul acumulatorului şi stochează rezultatul în acumulator.

1 0 0 0 0	S S S

Cicluri	:	1
Stări	:	4
Adresare	:	registru
Flaguri afectate	:	Z,S,P,CY,AC

Adună de la memorie:

Codul de operare	Operandul
ADD	M

Această instrucţiune adună conţinutul unei locaţii din memorie adresată de registrele H şi L, la conţinutul acumulatorului şi stochează rezultatul în acumulator. M este un simbol de referinţă la registrele H şi L.

1000 0110

Cicluri	:	2
Stări	:	7
Adresare	:	registru indirect
Flaguri afectate	:	Z,S,P,CY,AC

Exemple :

Se presupune că acumulatorul conţine 6CH şi registrul D conţine 2EH. Instrucţiunea ADD D efectuează adunarea în felul următor :

2EH = 00101110
6CH = 01101100

9AH = 10011010

Acumulatorul conţine valoarea 9AH după executarea instrucţiunii ADD D. Conţinutul registrului D rămîne neschimbat. Flagurile devin :

Z = 0, S = 1, P = 1, CY = 0, AC = 1

Următoarea instrucţiune dublează conţinutul acumulatorului : ADD A.

<u>ADI</u> <u>Adună imediat</u>

Instrucţiunea ADI adună conţinutul celui de al doilea bait al instrucţiunii la conţinutul acumulatorului şi stochează rezultatul în acumulator.

Codul de operare	Operandul
ADI	date

Operandul specifică datele reale ce trebuie să fie adunate la acumulator. Aceste date pot fi de forma unui număr, o constantă ASCII, eticheta unei valori definite anterior, sau o expresie. Datele nu pot depăşi un bait.

Asamblorul tratează toate simbolurile externe şi relocabile ca adrese de 16 biţi. Cînd unul din aceste simboluri este conţinut în expresia operand a unei instrucţiuni imdediate, el trebuie să fie precedat de unul din operatorii HIGH sau LOW care să specifice care bait al adresei va fi folosit în evaluarea expresiei. Cînd nu este dat operandul, asamblorul consideră un ope-

rator LOW şi dă un mesaj de eroare.

C6H =	1 1 0 0 0 1 1 0	ADI date
	Date	

Cicluri : 2
Stări : 7
Adresare : imediată.
Flaguri afectate: Z,S,P,CY,AC

Exemplu :

Se presupune că acumulatorul conţine valoarea 14 H.
Instrucţiunea ADI 66 are următorul efect :

Acumulatorul	= 14 H =	0001 0100
Datele directe	= 42 H =	0100 0010
		0101 0110 = 56 H

De notat că asamblorul converteşte valoarea zecimală 66 în valoare hexazecimală 42.

ANA SI logic cu acumulatorul

Instrucţiunea ANA realizează operaţiunea logică SI folosind baitul specificat şi acumulatorul, Rezultatul este plasat în acumulator.

Rezumat al operaţiunilor logice

Operaţiunea SI produce un rezultat 1, cînd cei doi biţi testaţi au şi unul şi celălalt valoarea 1.

Operaţiunea SAU produce un rezultat 1, cînd sau unu sau celălalt sau ambii testaţi au valoarea 1.

Operaţiunea SAU EXCLUSIV poriduce un bit 1 numai cînd cei doi biţi testaţi au valori diferite.

SI	SAU	SAU EXCLUSIV
1010 1010	1010 1010	1010 1010
0000 1111	0000 1111	0000 1111
0000 1010	1010 1111	1010 0101

SI logic între registru şi acumulator

Codul de operare	Operandul
ANA	reg.

Operandul trebuie să specifice unul din registrele A la E, H sau L. Instrucţiunea ANA realizează operaţiunea logică SI între conţinutul registrului specificat şi acumulator şi stochează rezultatul în acumulator. Transportul CY este resetat în zero.

A0H ... A5H sau A7H = | 1 0 1 0 0 | S S S | ANA reg

Cicluri	: 1
Stări	: 4
Adresare	: registru
Flaguri afectate	: Z,S,P,CY,AC

SI logic între memorie şi acumulator

Codul de operare	Operandul
ANA	M

Instrucţiunea realizează operaţiunea logică SI, între conţinutul locaţiei specificate din memorie şi acumulator, apoi stochează rezultatul în acumulator. Transportul CY este resetat în zero.

A6H = | 1 0 1 0 0 1 1 0 | ANA M

Cicluri	: 2
Stări	: 7
Adresare	: registru indirect
Flaguri afectate	: Z,S,P,CY,AC

Exemplu.

Operaţiunea logică SI efectuată între un bit avînd orice valoare şi un bit zero produce un zero.

Operaţiunea SI este folosită frecvent pentru a aduce în zero grupe de biţi. In exemplul următor, se realizează aducerea celor patru biţi cu pondere superioară ai acumulatorului în zero

iar cei patru biţi cu pondere inferioară rămîn neschimbaţi. S-a presupus că registrul C conţine OFH :

Acumulatorul	=	1 1 1 1	1 1 0 0	= OFCH
Registrul C	=	0 0 0 0	1 1 1 1	= OFH
		0 0 0 0	1 0 0 0	= OCH

ANI — SI imediat cu acumulatorul

Instrucţiunea ANI îndeplineşte operaţiunea logică SI folosind conţinutul celui de al doilea bait al instrucţiubii şi acumulatorul, Rezultatul este plasat în acumulator. Instrucţiunea ANI de asemenea resetează în zero transportul CY.

Codul de operare	Operandul
ANI	date

Operandul trebuie să specifice datele ce vor fi folosite în operaţiunea SI. Aceste date pot fi de forma unui număr, o constantă ASCII, o etichetă ce reprezintă o valoare definită anterior, sau o expresie. Datele nu pot depăşi un bait.

Asamblorul, tratează toate simbolurile externe şi relocabile ca adrese de 16 biţi. Cînd unul din aceste simboluri este conţinut în expresia operand a instrucţiunii imediate, ea trebuie să fie precedată de un operator HIGH sau LOW care să specifice care bait din adresă trebuie folosit pentru evaluarea expresiei. Cînd nu este dat nici-un operator asamblorul consideră un operator LOW şi trimite un mesaj de eroare.

E6H =	1 1 1 0 0 1 1 0	ANI date
	date	

Cicluri	: 2
Stări	: 7
Adresare	: imediată
Flaguri afectate:	Z,S,P,CY,AC

Exemplu :

Următoarea instrucţiune este folosită pentru a reseta (în zero) bitul 6 din baitul conţinut în acumulator :

ANI 10111111B

Fiind cunoscut că operaţiunea logică SI efectuată între un bit avînd orice valoare şi un bit zero produce un zero, se poate vedea că instrucţiunea de mai sus va face ca numai bitul 6 să fie schimbat, ceilalţi biţi 0 ... 5 şi 7 rămînînd neschimbat, Această tehnică este frecvent folosită cînd programul foloseşte biţii individuali dintr-un bait cu indicatori de stare.

<u>CALL</u> <u>Asocierea instrucţiunilor PUSH SI JMP</u>

Instrucţiunea CALL combină funcţiile instrucţiunilor PUSH şi JMP. Instrucţiunea CALL introduce în stivă conţinutul contorului de program (adresa următoarei instrucţiuni secvenţiale) şi apoi face un salt la adresa specificată în instrucţiunea CALL.

Fiecare instrucţiune CALL sau una din variantele sale implică utilizarea unei instrucţiuni RET (întoarcere).

Cal de operare	Operandul
CALL	dresa

Adresa poate fi specificată printr-un număr, o etichetă sau o expresie.(Eticheta e cea mai utilizază). Asamblorul inversează baitul superior şi cel inferior al adresei cînd asamblează instrucţiunea.

1 1 0 0 1 1 0 1
baitul inferior adresă
baitul superior adresă

Cicluri	:	5
Stări	:	17 (18 la 8085)
Adresare	:	imediată/registru indirect
Flaguri adectate	:	nu

Exemplu :

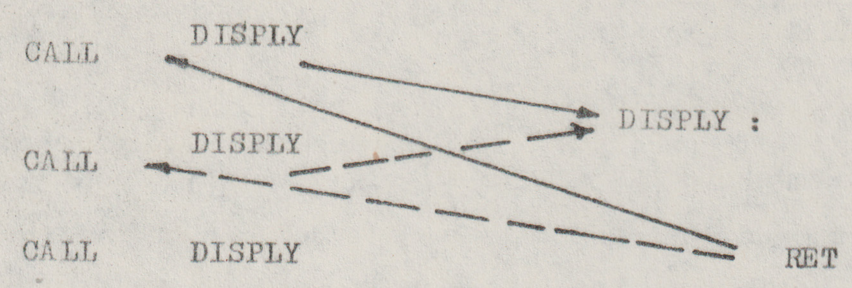
Cînd o secvenţă dată este necesară de mai multe ori într-un program, se poate economisii memoria prin codarea acelei secvenţe ca subrutină refolosibilă cu ajutorul instrucţiunii CALL sau a uneia din variantele acesteia. De exemplu, să presupunem că într-o aplicaţie este comandat un display de 6 cifre cu LED-uri.Rezulta-

tele a diferite calcule trebuie să fie afişate pe display de mai multe ori în cursul unui program, ceea ce ar necesita ca display-ul să fie trecut ON LINE în diverse puncte ale programului. Practic, se face o codare ca subrutină. Dacă se asignează o etichetă DISPLY pentru prima instrucţiune de comandă a display-ului,următoarea instrucţiune CALL poate adresa subrutina display

CALL DISPLY

Instrucţiuhea CALL introduce în stivă adresa următoarei instrucţiuni din program şi apoi transferă controlul la subrutina DISPLY.

Subrutina DISPLY trebuie apoi să execute o instrucţiune de întoarcere (sau una din variantele acesteia) la desfăşurarea normală a programului, Următorul grafic ilustrează efectul instrucţiunii CALL şi a instrucţiunii de retur.



Consideraţiuni pentru folosirea subrutinei

Segmente de cod mai mari pot fi repetate ori de cîte ori se doreşte în cadrul unui program, folosindu-se subrutinele care economisesc importante spaţii în memorie. Astfel, dacă în exemplul dat mai înainte, comanda displey-ului ar necesita 100 byte-ţi şi ar fi necesar ca aceasta să se repete de 3 ori în cadrul programului, s-ar folosi 300 byte-ţi. Dacă se utilizează o subrurină sînt necesari numai 100 byte-ţi plus nouă baiţi pentru trei instrucţiuni CALL.

De notat că subrutinele necesită folosirea unei stive. Practic este necesar să se prevadă o memorie RAM pentru stivă.

CC — CALL dacă CY = 1

Instrucţiunea CC combină funcţiunile instrucţiunilor JC şi PUSH. Instrucţiunea CC testează indicatorul CY. Dacă CY = 1 instrucţiunea CC introduce în stivă conţinutul contorului de program şi face un salt la adresa specificată în baiţii doi şi trei ai instrucţiunii CC. Dacă CY = 0 executarea programului continuă cu următoarea instrucţiune secvenţială.

Codul de operare	Operandul
CC	adresa

Cu toate că folosirea unei etichete este mai uzuală, adresa poate, de asemenea, să fie specificată printr-un număr sau o expresie.

DCH =	1 1 0 1 1 1 0 0	CC addr
	bitul inferior adresă	
	bitul superior adresă	

Cicluri : 3 sau 5 (2 sau 5 la 8085)
Stări : 11 sau 17 (9 sau 18 la 8085)
Adresare : directă/registru indirect
Flaguri afectate: nu

CM — CALL dacă semnul este minus

Instrucţiunea CM combină funcţiile instrucţiunilor JM şi PUSH. Instrucţiunea CM testează flagul S. Dacă S = 1 (indicând că conţinutul acumulatorului are valoare negativă), instrucţiunea CM introduce în stivă conţinutul contorului de program şi face un salt la adresa specificată de instrucţiunea CM. Dacă S = 0 executarea programului continuă cu următoarea instrucţiune secvenţială.

Codul de operare	Operandul
CM	adresa

Cu toate că folosirea unei etichete este mai utilizată, adresa poate de asemenea să fie specificată printr-un număr sau o expresie.

FCH =

1 1 1 1 1 1 0 0
baitul inferior adresă
baitul superior adresă

Cicluri : 3 sau 5 (2 sau 5 la 8085)
Stări : 11 sau 17 (9 sau 18 la 8085)
Adresare: imediat/registru indirect
Flaguri afectate : nu

CMA Complement acumulator

Instrucţiunea CMA completează fiecare bait al acumulatorului obţinîndu-se astfel complementul de unu. Toate flagurile rămîn neschimbate.

Codul de operare	Operandul
CMA	

Operandul nu este permis cu instrucţiunea CMA.

2FH = | 0 0 1 0 1 1 1 1 | CMA

Cicluri: 1
Stări : 4
Flaguri afectate: nu

Pentru a se obţine complementul de doi se adună 1 la conţinutul acumulatorului, după ce a fost executată instrucţiunea CMA.

Exemplu :

Presupunînd că acumulatorul conţine valoarea 51H, după ce a fost executată instrucţiunea CMA, în acumulator va fi valoarea OAEH.

51H = 01010001
OAEH = 10101110

CMC Complement CY

Dacă CY = 0 instrucţiunea CMC provoacă schimbarea CY=1. Dacă CY=1 instrucţiunea CMC provoacă schimbarea CY=0. Toate ce-

lelalte flaguri rămîn neschimbate.

Codul de operare	Operandul
CMC	

Operandul nu este permis în instrucţiunea CMC.

3FH = | 0 0 1 1 1 1 1 1 | CMC

Cicluri : 1
Stări : 4
Flaguri afectate : numai CY

Exemple :

Să presupunem că programul foloseşte bitul 7 al baitului pentru a controla dacă subrutina este apelată. Pentru a testa bitul respectiv, programul încarcă baitul în acumulator, roteşte bitul 7 în flagul CY şi execută instrucţiuhea CC (CALL dacă CY=1).

Inainte de a se reveni la program, subrutina reiniţializează baitul flag folosind următorul cod :

```
CMC     ; Complementează CY
RAR     ; Adu bitul 7 în 0 prin rotare dreaptă
RET     ; RETURN
```

CMP Compară cu acumulatorul

Instrucţiunea CMP compară baitul specificat cu conţinutul acumulatorului şi indică rezultatul prin setarea flagului Z şi CY. Valorile ce au fost comparate rămîn neschimbate. Z = 1 indică egalitatea. CY = 0 arată că acumulatorul este mai mare sau egal cu baitul specificat. CY = 1 arată că acumulatorul este mai mic ca baitul. Oricum semnificaţiile lui CY sînt inversate cînd valorile au semne diferite sau una din valori este complementată.

Programul testează flagurile folosind una din instrucţiunile JUMP codiţionat CALL sau RETURN.

De exemplu : JZ (JUMP dacă este zero) testează egalitatea.

Descriere funcţională

Compararea este făcută prin scăderea baitului specificat din conţinutul acumulatorului, după care flagurile Z şi CY vor indica rezultatul. Această scădere foloseşte registrele interne

ale procesorului. Deorece scăderea se efectuează prin adunarea complementului de doi,instrucţiunea CMP recomplementează CY-ul generat de scădere.

Compară registrul cu acumulatorul

Codul de operare	Operandul
CMP	reg

Operandul trebuie să numească unul din registrele A la E,H sau L.

E8H ... BDH sau NFH = | 1 0 1 1 1 | S S S | CMP reg

Cicluri : 1
Stări : 4
Adresare: registru
Flaguri afectate: Z,S,P,CY,AC

Concluzie

A = R	Z = 1		A R	CY = 1	
A R	Z = 0		A R	CY = 0,2 = 0	
A R	CY= 0		A = conţinut acumulator		
			R = conţinut registru		

Compară memoria cu acumulatorul

Codul de operare	Operandul
CMP	M

Această instrucţiune compară conţinutul locaţiei din memorie adresată de registrele H şi L, cu conţinutul acumulatorului.M este un simbol de referinţă la registrele pereche H şi L.

BEH = 1 0 1 1 1 1 1 0 CMP M

Cicluri : 2
Stări : 7
Adresare : registru indirect
Flaguri adecvate : Z,S,P,CY,AC

Exemplul 1

Se presupune că acumulatorul conţine valoarea OAH şi registrul E conţine valoarea 05 H. Instrucţiunea CMP E îndeplineşte următoarea scădere internă (se reaminteşte că scăderea este în realitate adunarea în complement de doi).

Acumulatorul =	0 0 0 0 1 0 1 0	
+(-Registru E) =	1 1 1 1 1 0 1 1	
	0 0 0 0 0 1 0 1	+(-CY)

După ce CY este complementat datorită operaţiunii de scădere, ambii biţi Z şi CY au valoarea zero, ceea ce înseamnă că A este mai mare ca E.

Exemplul 2

Se presupune că acumulatorul conţine valoarea - 1 BH şi registrul E conţine 05 H :

Acumulatorul =	1 1 1 0 0 1 0 1	
+(-Registrul E)=	1 1 1 1 1 0 1 1	
	1 1 1 0 0 0 0 0	+(-CY)

După ce instrucţiunea CMP recomplementează pe CY ambii biţi Z şi CY au valoarea zero. Normal aceasta arată că acumulatorul A este mai mare ca registrul E. Dacă se ţine însă seama că acumulatorul conţine o valoare negativă, trebuie ca interpretarea să fie inversată, deci A E. Sarcina interpretării corecte o are programul.

CNC CALL dacă CY = 0

Instrucţiunea CNC combină funcţiunile instructiunilor JNC şi PUSH. Instrucţiunea CNC testează bitul CY. Dacă CY = 0 instrucţiunea CNC introduce în stivă conţinutul contorului de program şi face un salt la adresa indicată de instrucţiunea CNC. Dacă CY = 1, executarea programului continuă cu următoarea instrucţiune secvenţială.

Codul de operare	Operandul
CNC	adresa

In general se foloseşte o etichetă, adresa însă poate de asemenea să fie un număr sau o expresie.

D4H =	1 1 0 1 0 1 0 0	CNC addr
	baitul inferior adresă	
	baitul superior adresă	

Cicluri : 3 sau 5 (2 sau 5 la 8085)
Stări : 11 sau 17 (9 sau 18 la 8085)
Adresare :imediată/registru indirect
Flaguri afectate;nu

CNZ **CALL dacă Z = 1**

Instrucţiunea CNZ combină funcţiunile instrucţiunilor JNZ şi PUSH. Instrucţiunea CNZ testează bitul Z. Dacă Z = 0 (indicînd că conţinutul acumulatorului este diferit de zero) ins - trucţiunea CNZ introduce în stivă conţinutul contorului de program şi face un salt la adresa specificată de instrucţiunea CNZ. Dacă Z = 1, executarea programului continuă cu următoarea instrucţiune secvenţială.

Codul de operare	Operandul
CNZ	adresa

In general se foloseşte o etichetă, adresa poate însă să fie un număr sau o expresie.

C4H =	1 1 0 0 0 1 0 0	CNZ addr
	baitul inferior adresă	
	baitul superior adresă	

Cicluri : 3 sau 5 (2 sau 5 la 8085)
Stări : 11 sau 17 (9 sau 18 la 8085)
Adresare : imediată/registru indirect
Flaguri afectate : nu

CP **CALL dacă S = 0**

Instrucţiunea CP combină funcţiunile instrucţiunilor JP şi PUSH. Instrucţiunea CP testează bitul S. Dacă S = 0 (indicînd că conţinutul acumulatorului este pozitiv), instrucţiunea CP introduce în stivă conţinutul contorului de program şi face un salt la adresa specificată de instrucţiunea CP. Dacă S = 1, executarea programului continuă cu următoarea instrucţiune secvenţială.

Codul de operare	Operandul
CP	adresa

In general se folosește o etichetă, adresa poate fi de asemenea un număr sau o expresie.

F4G =

1	1	1	1	0	1	0	0
baitul inferior adresă							
baitul superior adresă							

CP addr

Cicluri : 3 sau 5 (2 sau 5 la 8085)
Stări : 11 sau 17 (9 sau 18 la 8085)
Adresare: imediat/registru indirect
Flaguri afectate : nu

CPE CALL dacă P = 1 (bait par)

Partitatea P a baitului din acumulator este pară, cînd numărul de biți "1" din bait este par. Această condiție este indicată cînd P = 1.

Instrucțiunile CPE și CPO sînt folosite pentru testarea parității datelor la intrare. Oricum instrucțiunea IN nu schimnici-unul din indicatorii de condiții.

Instrucțiunea CPE combină funcțiile instrucțiunilor JPE și PUSH. Instrucțiunea CPE testează bitul P. Dacă P = 1, instrucțiunea CPR introduce în stivă conținutul contorului de program și face un salt la adresa specificată de instrucțiunea CPE. Dacă P=0, executarea programului continuă cu următoarea instrucțiune secvențială.

Codul de operare	Operandul
CPE	adresa

In general se folosește o etichetă, adresa poate însă să fie un număr sau o expresie.

ECH =

1	1	1	0	1	1	0	0
baitul inferior adresă							
baitul superior adresă							

CPE addr

Cicluri : 3 sau 5 (2 sau 5 la 8085)
Stări : 11 sau 17 (9 sau 17 la 8085)
Adresare : direct/registru indirect
Flaguri afectate : nu

CPI   Compară imediat

Instrucţiunea CPI comportă conţinutul celui de al doilea bait al său cu conţinutul acumulatorului şi setează flagurile Z şi CY pentru a indica rezultatul. Valorile comparate rămîn neschimbate.

Flagul Z indică egalitatea. Lipsa transportului ,(CY=0) arată că conţinutul acumulatorului este mai mare ca valoarea datelor directe. De reţinut că, semnificaţia flagului CY este inversată cînd valorile au semne diferite sau una din valori este complementată.

Codul de operare	Operandul
CPI	date

Operandul trebuie să specifice datele ce trebuie să fie comparate. Acestea pot fi de forma unui număr, o constantă ASCII, o etichetă a cărei valoare a fost definită anterior, sau o expresie. Datele nu pot depăşi un bait.

Asamblorul se caracterizează prin tratarea tuturor simbolurilor externe şi relocabile ca adrese de 16 biţi. Cînd unul din aceste simboluri este conţinut în expresie operand a instrucţiunii directe, ea trebuie să fie precedată de un operator HIGH sau LOW, care să specifice care bait din adresă trebuie folosit pentru evaluarea expresiei. Cînd nu este dat nici-un operator asamblorul consideră un operator LOW şi este dat un mesaj de eroare.

FEH =	1	1	1	1	1	1	1	0	CPI date
	date								

Cicluri : 2
Stări : 7
Adresare: registru indirect
Flaguri afectate : Z,S,P,CY,AC

Exemplu :

Instrucţiunea CPI ´C ´compară conţinutul acumulatorului cu litera C = 43 H.

CPO CALL dacă P = O (ait impar)

Paritatea P a baitului din acumulator este impară, cînd numărul de biţi "1" din bait este impar. Bitul de paritate are valoarea P = O cînd este existentă această condiţie.

Instrucţiunile CPO şi CPE sînt folosite pentru testarea parităţii datelor la intrare. Oricum, instrucţiunea IN nu setează nici unu din flaguri.

Instrucţiunea CPO combină funcţiile instrucţiunilor JPO şi PUSH. Instrucţiunea CPO testează bitul P. Dacă P = O, instrucţiunea CPO introduce în stivă conţinutul contorului de program şi face un salt la adresa specificată în instrucţiune. Dacă P = 1, executarea programului continuă cu următoarea instrucţiune secvenţială.

Codul de operare	Operandul
CPO	adresa

In general se foloseşte o etichetă, adresa poate să fie de asemenea un număr sau o expresie.

E4H =	1 1 1 0 0 1 0 0	CPO addr
	baitul inferior adresă	
	baitul superior adresă	

Cicluri : 3 sau 5 (2 sau 5 la 8085)
Stări : 11 sau 17 (9 sau 18 la 8085)
Adresare: direct/registru indirect
Flag: nu

CZ CALL dacă Z = 1

Instrucţiunea CZ combină funcţiunile instrucţiunilor JZ şi PUSH, Instrucţiunea CZ testează bitul Z. Dacă Z = 1 (adică conţinutul acumulatorului are valoarea zero) instrucţiunea CZ introduce în stivă conţinutul contorului de program şi face un

salt la adresa specificată în instrucţiune.

Dacă Z = 0 (adică conţinutul acumulatorului are o va - loare diferită de zero), programul continuă executînd următoarea instrucţiune.

Codul de operare	Operandul
CZ	adresa

In general se foloseşte o etichetă, adresa însă poate de asemenea să fie un număr sau o expresie.

CCH =	1 1 0 0 1 1 0 0	CZ addr
	baitul inferior adresă	
	baitul superior adresă	

Cicluri : 3 sau 5 (2 sau 5 la 8085)
Stări : 11 sau 17 (9 sau 18 la 8085)
Adresare: direct/registru indirect
Flaguri adectate : nu

DAA Schimbă valoarea acumulatorului în zecim

Instrucţiunea DAA modifică valoarea celor opt biţi din acumulator sub forma a două grupe de cîte patru biţi codate în zecimal.

Codul de operare	Operandul
DAA	

In instrucţiunea DAA nu este admis nici un operand. Instrucţiunea DAA se foloseşte cînd se face o adunare între numere zecimale. Aceasta este singura instrucţiune care necesită folosirea unui transport auxiliar (AC).

Instrucţiunea DAA, practic, se foloseşte imediat după o instrucţiune aritmetică.

Instrucţiunea DAA operează în felul următor:

1. Dacă ultimii patru biţi de pondere inferioară din acumulator au o valoare mai mare decît 9, sau dacă flagul AC=1, instrucţiunea DAA adună 6 la acumulator.

2. Dacă primii patru biţi de pondere superioară din acumulator au o valoare mai mare de cît 9, sau dacă flagul AC=1, instrucţiunea DAA adună 6 la cei patru biţi de pondere superioară ai acumulatorului.

27H = | 0 0 1 0 0 1 1 1 | DAA

Cicluri : 1
Stări : 4
Adresare: registru
Flaguri afectate: Z,S,P,CY,AC

Exemple :

Se presupune că acumulatorul conţine valoarea 9BH rezultată din adunarea 08H plus 93H :

CY	AC	
0	0	
1001	0011	= 93 H
0000	1000	= 08 H
1001	1011	= 9 BH

Deoarece, ultimii patru biţi de pondere inferioară ai conţinutului 9 BH, din acumulator au o valoare mai mare ca 9 , (B 9), instrucţiunea adună 6 la conţinutul acumulatorului :

CY	AC	
0	1	
1001	1011	= 9 BH
0000	0110	= 06 H
1010	0001	A1H

Acum primii patru biţi de pondere superioară ai conţinutului A1H din acumulator au o valoare mai mare ca 9, (A 9) şi instrucţiunea adună 6 la acest grup de patru biţi :

CY	AC	
1	1	
1010	0001	= A1H
0110	0000	= 60H
0000	0001	= 101 = 08 + 93

După terminarea instrucţiunii DAA acumulatorul conţine valoarea 01 în cod BCD, ambele flaguri: CY şi AC avînd valoarea 1. Deoarece rezultatul real al adunării e 101, flagul carry poate avea semnificaţie pentru program.

DAD — Adunarea registrelor duble

Instrucţiunea DAD adună valoarea celor 16 biţi dintr-o pereche de registre specificate cu conţinutul registrelor pereche H şi L. Rezultatul este stocat în H şi L.

Codul de operare	Operandul
DAD	{B, D, H, SP}

Instrucţiunea DAD poate aduna numai conţinutul unei perechi de registre B şi C, D şi E, H şi L. sau SP (Stack Pointer-indicator de stivă) la conţinutul lui H şi K, De notat că registrul pereche H şi L este specificat în instrucţiune deoarece poate fi adunat cu el însăşi.

Instrucţiunea DAD modifică indicatorul CY dacă exista un transport de la registrele H şi L. Instrucţiunea DAD nu afectează nici unu din indicatori în afară de CY.

09 H =	0 0	0 0	1 0 0 1	DAD B
19 H =	0 0	0 1	1 0 0 1	DAD D
29 H =	0 0	1 0	1 0 0 1	DAD H
39 H =	0 0	1 1	1 0 0 1	DAD SP

Cicluri : 3
Stări : 10
Adresare: registru
Flaguri afectate : CY

Exemple :

Instrucţiunea DAD prevăzută pentru a salva conţinutul registrului SP.

```
LXI,H, OOH      ; aduce H&L la zero
DAD SP          ; Introduce conţinutul SP în H&L
SHLD SAVSP      ; STORE SP IN MEMORY
```

Instrucţiunea DAD H dublează numărul conţinut în registrele H şi L cu excepţie cînd operaţiubea provoacă un transport din registrul H.

DCR Decrement

Instrucţiunea DCR scade cifra 1 din conţinutul baitului specificat.

Instrucţiunea DCR afectează toate flagurile cu excepţia lui CY. Deorece instrucţiunea DCR conservă flagul CY ea poate fi folosită în rutinele aritmetice multibait pentru decrementarea numerelor sau alte sarcini similare.

Decrement registru

Codul de operare	Operandul
DCR	reg

Operandul trebuie să specifice unul din registrele A pînă la E, H sau L. Instrucţiunea scade cifra 1 din caloarea conţinutului specificat.

Cod		Instrucţiune
05HE	= | 0 0 | D D D | 1 0 1 |	DCR B
0DH		DCR C
15H	Cicluri : 1	DCR D
1DH	Stări : 5(4 la 8085)	DCR E
25H	Adresare: registru	DCR H
2DH	Flaguri afectate:Z,S,P,AC	DCR L
3DH		DCR A

Decrement memorie

Codul de operare	Operandul
DCR	M

Instrucţiunea scade cifra 1 din valoarea conţinutului locaţiei din memorie adresată de registrele H şi L. M este un simbol de referinţă la baitul din memorie adresat de registrele H şi L.

35 H : | 0 0 1 1 0 1 0 1 | **DCR M**

Cicluri : 3

Stări : 10

Adresare : registru indirect

Flaguri afectate : Z,S,P,AC

Exemplu :

Instrucţiunea DCR este frecvent folosită pentru operaţiuni de control multibait cum ar fi mutarea unui număr de caractere dintr-o arie a memoriei intr-alta.

```
      MVI B, 5H      ; set contorul de control
      LXI H,250H     ; încarcă H şi L cu adresa sursă
      LXI D,900H     ; încarcă D şi E cu adresa destinaţie
LOOP: MOV A,M        ; încarcă baitul ce trebuie mutat
      STAX D         ;stochează baitul
      DCX D          ; decrementează adresa destinaţie
      DCX H          ; decrementează adresa sursă
      DCR B          ; decrementează contoţul de control
      XRA A          ; sterge acumulatorul
      CMP B          ; compară contorul de control cu zero
      JNZ LOOP       ; mută alt bait dacă contorul nu este zero.
```

<u>DCX</u> <u>Decrement registru pereche</u>

Instrucţiunea DCX scade cifra 1 din valoarea conţinutului registrelor pereche specifice. Instrucţiunea DCX nu afectează nici una din flagúri. Deorece instrucţiunea DCX conservă toate flagurile ea poate fi folosită pentru modificarea adresei în orice secvenţă de instrucţiuni.

Codul de operare	Operandul
DCX	{B, D, H, SP}

Instrucţiunea DCX nu decrementează decît registrele pereche B şi C, D şi E, H şi L, sau SP, notaţiile B,D,H sînt specificate pentru a indica registrele pereche de mai sus.

Aveţi grijă la decrementarea SP pentru că provoacă pierderea sincronizării între indicator şi conţinutul real al stivei.

0BH =	0 0	0 0	1 0 1 1	DCX B
1BH =	0 0	0 1	1 0 1 1	DCX D
2BH =	0 0	1 0	1 0 1 1	DCX H
3BH =	0 0	1 1	1 0 1 1	DCX SP

Cicluri : 1
Stări : 5 (6 la 8085)
Adresare: registru
Flaguri afectate: nu

Exemplu :

Se presupune că registrele H şi L conţin adresa 9800 H cînd este executată instrucţiunea DCX H.

Instrucţiunea DCX consideră conţinutul celor două registre ca o singură valoare de 16 biţi şi face un împrumut de la registrul H (registrul L avînd valoarea zero) pentru a rezulta valoarea 97FFH.

DI Intreruperi inactivate

Sistemul de întreruperi este inactivat cînd procesorul recunoaşte o întrerupere sau imediat după executarea unei instrucţiuni DI.

In aplicaţiile care folosesc întreruperi, instrucţiunea DI este frecvent folosită numai cînd nu trebuie să fie întreruptă o secvenţă de cod. De exemplu, secvenţele de cod dependente de timp devin inexacte dacă sînt întrerupte. Pentru a se evita această incorectitudine întreruperile pot fi inactivate, prin introducerea unei instrucţiuni DI la începutul secvenţei de cod respective. Deoarece nu se poate prevedea apariţia unei întreruperi, se va prevedea după secvenţa de cod dependenţă de

timp, care nu trebuie să fie întreruptă, o instrucţiune EI.

Codul de operare	Operandul
DI	

Cu instrucţiunea DI nu este permis nici-un operand.

F3H = | 1 1 1 1 0 0 1 1 | DI

Cicluri : 1
Stări : 4
Flaguri afectate : nu

EI Intreruperi activate

Instrucţiunea EI activează sistemul de întreruperi după care se trece la executarea următoarei instrucţiuni din program.

In aplicaţiile care folosesc întreruperi, sistemul de întreruperi este frecvent inactivat numai cînd procesorul acceptă o întrerupere sau cînd secvenţa de cod nu trebuie să fie întreruptă. Sistemul de întreruperi se poate inactiva prin introducerea unei instrucţiuni DI la începutul secvenţei de cod iar din cauză că nu poate fi prevăzută apariţia întreruperii se va include o instrucţiune EI la sfîrşitul secvenţei de cod.

Codul de operare	Operandul
EI	

Cu instrucţiunea EI nu este permis nici-un operand.

FBH = | 1 1 1 1 1 0 1 1 | EI

Cicluri : 1
Stări : 4
Flaguri afectate : nu

Exemplu :

Instrucţiunea EI este frecvent folosită în secvenţa de pornire. Cînd eliminarea e conectată microcalculatorul începe să opereze la o serie de adrese nedeterminate. Aplicarea unui semnal RESET forţează contorul de program să treacă în zero .

O secvenţă comună de instrucţiuni în acest punct este EI,HLT. Aceste instrucţiuni activează sistemul de întreruperi (RESET de asemenea desactivează sistemul de întreruperi) şi opreşte procesorul.

HLT Oprire

Instrucţiunea HLT opreşte procesorul, In momentul opririi contorul programului conţine adresa următoarei instrucţiuni secvenţiale iar flagurile şi registrele rămîn neschimbate.

76 H =	0	1	1	1	0	1	1	0	HLT

Cicluri : 1

Stări : 7 (5 la 8085)

Flaguri afectate : nu

Odată oprit procesorul poate fi restartart numai de un eveniment extern, de regulă o întrerupere. Dacă în microprocesorul 8080 este executată instrucţiunea HLT cînd întreruperile sînt inactive, singura posibilitate de a restructura procesorul este prin aplicarea semnalului RESET.

Acesta forţează contorul programului în zero. Proceso - rul poate temporar să abandoneze starea de oprire dacă este cerut serviciul de acces direct la memorie. Procesorul reintră în starea halt după ce cererea respectivă a fost satisfăcură.

IN Intrarea de la periferice

Instrucţiunea IN citeşte opt biţi de date de la portul specificat şi îi încarcă în acumulator.

Codul de operare	Operandul
IN	expresie

Expresia operandului poate fi un număr sau orice expresie care corespunde unei valori cuprinse în domeniul OOH la OFFH.

DBH =	1 1 0 1 1 0 1 1	IN
	expresia	

Cicluri : 3
Stări : 10
Adresare: directă
Flaguri afectate : nu

INR Increment

Instrucţiunea INR adună cifra 1 la conţinutul baitului specificat. Această instrucţiune afectează toate flagurile cu excepţia flagului CY.

Deoarece instrucţiunea INR conservă pe CY ea poate fi folosită în rutinele aritmetice multibait pentru incrementarea contorului de caractere şi pentru alte scopuri similare.

Increment registru

Codul de operare	Operandul
INR	registru

Operandul trebuie să specifice unul din registrele A la E, H sau L. Instrucţiunea adună cifra 1 la conţinutul registrului specificat.

Cod		Instrucţiune
04H	= | 0 0 | D D D | 1 0 0 |	INR B
0CH		INR C
14H	Cicluri : 1	INR D
1CH	Stări : 5 (4 la 8085)	INR E
24H	Adresare : registru	INR H
2CH	Flaguri afectate : Z,S,P,AC	INR L
3CH		INR A

Increment memorie

Codul de operare	Operandul
INR	M

Această instrucţiune incrementează cu 1 conţinutul locaţiei din memorie adresată de registrele H şi L.

M reprezintă un simbol de referire la registrele H şi L.

34 H = | 0 0 1 1 0 1 0 0 | INR M

Cicluri : 3
Stări : 10
Adresare: registru indirect
Flaguri afectate : Z,S,P,AC

Exemplu :

Dacă registrul C conţine 99H, instrucţiunea INR C incrementează conţinutul acestui registru, care devine 9AH.

INX — Increment registru pereche

Instrucţiunea INX adună 1 la conţinutul perechilor de registre specificate. Această instrucţiune nu afectează niciu un din flaguri. Deoarece instrucţiunea INX conservă toate flagurile ea poate fi folosită pentru modificarea adreselor în rutinele aritmetice multibait.

Codul de operare	Operandul
INX	{B, D, H, SP}

Instrucţiunea INX poate incrementa numai perechea de registre B şi C, D şi E, H şi L sau SP (Stack Pointer). De notat că literele B,D,H şi SP trebuie să specifice perechile de registre mai sus menţionate.

Acţionaţi cu multă atenţie cînd se încrementează indicatorul de stivă.

Se presupune, de exemplu că instrucţiunea INX SP este executată după ce un număr de byte-ţi au fost introduşi în stivă. Instrucţiunea POP (extragere din stivă) accesează baitul de ordine superioară al celei mai recente intrări în stivă şi baitul de ordine inferioară al următoarei intrări mai vechi în stivă. Similar, o instrucţiune OUSH adună doi biţi noi la stivă dar SP va indica baitul de ordine inferioară al acestei noi intrări de byte-ţi.

03H	=	0 0	0 0	0 0 1 1	INX B
13H	=	0 0	0 1	0 0 1 1	INX B
23H	=	0 0	1 0	0 0 1 1	INX H
33H	=	0 0	1 1	0 0 1 1	INX SP

Cicluri : 1
Stări : 5 (6 la 8085)
Adresare: registru
Flaguri afectate : nu

Exemple :

Se presupune că registrele D şi E conţin valoarea 01FFH. Instrucţiunea INX D incrementează valoarea registrelor obţinîndu-se 0200 H. Dacă s-ar fi aplicat instrucţiunea INE E care ignoră transportul (CY=1) rezultat de la baitul de ordine inferioară s-ar obţine ca rezultat 0100 H. (Această condiţie ar putea fi detectată prin testarea indicatorului de condiţie Z).

Dacă registrul SP conţine valoarea OFFFFH, instrucţiunea INX SP incrementează conţinutul registrului SP care devine 0000H. Instrucţiunea INX nu modifică nici un flag pentru a indica această condiţie.

JC Salt dacă există transport

Instrucţiunea JC testează modificarea condiţiei CY. Dacă CY = 1, programul execută un salt la adresa indicată în instrucţiunea JC. Dacă CY = 0 programul continuă cu executarea următoarei instrucţiuni secvenţiale.

Codul de operare	Operandul
JC	adresa

Adresa poate fi specificată printr-un număr, o etichetă sau o expresie. Asamblorul inversează baiţii de ordine superioară şi inferioară ai adresei cînd asamblează instrucţiunea.

DAH	=	1 1 0 1 1 0 1 0	JC addr
		baitul inferior adresă	
		baitul superior adresă	

Cicluri : 3(2 sau 3 la 8085)
Stări : 10(7 sau 10 la 8085)
Adresare : imediată
Flaguri afectate : nu

Exemple cu variante ale instrucţiunii de salt sînt date la descrierea instrucţiunii JPO.

JM Salt dacă este minus

Instrucţiunea JM testează indicatorul de semn S. Dacă conţinutul acumulatorului este negativ (S=1), programul execută un salt la adresa specificată în instrucţiunea JM. Dacă conţinutul acumulatorului este pozitiv (S=0), programul continuă cu executarea următoarei instrucţiuni secvenţiale.

Codul de operare	Operandul
JM	adresa

Adresa poate fi specificată ca un număr, o etichetă sau o expresie. Asamblorul inversează baiţii (superior şi inferior) adresei cînd asamblează instrucţiunea.

FAH	1 1 1 1 1 0 1 0	JM addr
	baitul inferior adresă	
	baitul superior adresă	

Cicluri : 3 (2 sau 3 la
Stări : 10 (7 sau 10 la 8085)
Adresare: imediată
Flaguri afectate : nu

JMP Salt

Instrucţiunea JMP schimbă executarea unei secvenţe de program, încărcînd adresa celui de al doilea şi al treilea bait al său în contorul de program.

Codul de operare	Operandul
JMP	adresa

Adresa poate fi specificată ca un număr, o etichetă sau o expresie. Asamblorul inversează baiţii (superior cu inferior) ai adresei cînd asamblează adresa.

C3H

1 1 0 0 0 0 1 1
baitul inferior adresă
baitul superior adresă

JMP addr

Cicluri : 3
Stări : 10
Adresare : imediată
Flaguri afectate : nu

<u>JNC</u> <u>Salt dacă nu exidtă transport</u>

Instrucţiunea JNC testează indicatorul CY de transport) Dacă nu există transport (CY=0) programul execută un salt la adresa specificată în instrucţiune JNC. Dacă există un transport (CY=1) programul continuă executarea următoarei instrucţiuni secvenţiale.

Codul de operare	Operandul
JNC	adresa

Adresa poate fi specificată ca un număr, o etichetă sau o expresie. Asamblorul inversează baiţii (superior şi inferior) ai adresei apoi asamblează instrucţiunea.

D2H =

1 1 0 1 0 0 1 0
baitul inferior adresă
baitul superior adresă

JNC addr

Cicluri : 3 (2 sau 3 la 8085)
Stări : 10 (7 sau 10 la 8085)
Adresare: imediată
Flaguri afectate : nu

JNZ — Salt dacă nu este zero

Instrucţiunea JNZ testează flagul Z (zero). Dacă conţinutul acumulatorului este diferit de zero (Z=0), programul execută un salt la adresa specificată în instrucţiunea JNZ. Dacă conţinutul acumulatorului este zero (Z=1), execuţia continuă cu următoarea instrucţiune secvenţială .

Codul de operare	Adresa
JNZ	adresa

Adresa poate fi specificată ca un număr, o etichetă sau o expresie. Asamblorul inversează baiţii (superior şi inferior) ai adresei cînd asamblează instrucţiunea.

02H =	1 1 0 0 0 0 1 0	JNZ addr
	baitul inferior adresă	
	baitul superior adresă	

Cicluri : 3 (2 sau 3 la 8085)
Stări : 10 (7 sau 10 la 8085)
Adresare: imediată
Flaguri afectate : nu

JP — Salt dacă este plus

Instrucţiunea JP testează indicatorul S (de semn). Dacă conţinutul acumulatorului este pozitiv (S = 0), programul execută un salt la adresa specificată în instrucţiunea JP. Dacă conţinutul acumulatorului este negativ (S=1), programul continuă cu executarea următoarei instrucţiuni secvenţiale.

Codul de operare	Operandul
JP	adresa

Adresa poate fi specificată ca un număr, o etichetă sau o expresie. Asamblorul inversează baiţii (superior şi inferior) ai adresei, cînd asamblează instrucţiunea.

F2H =

1 1 1 1 0 0 1 0
baitul inferior adresă
baitul superior adresă

JP addr

Cicluri : 3 (2 sau 3 la 8085)
Stări : 10 (7 sau 10 la 8085)
Adresare: directă
Flaguri afectate : nu

JPE Salt dacă paritatea este pară

Paritatea este pară dacă baitul din acumulator conţine un număr par de biţi "1". In acest caz Z = 1.

Instrucţiunea JPE testează indicatorul P. Dacă numărul de biţi 1 este par (P=1) programul execută un salt la adresa specificată în instrucţiunea JPE. Dacă numărul de biţi 1 este impar (P=0) programul continuă executarea următoarei instrucţiuni secvenţiale.

Codul de operare	Operandul
JPE	adresa

Adresa poate fi specificată ca un număr, o etichetă sau o expresie. Asamblorul inversează baiţii (superior şi inferior) ai adresei, cînd asamblează instrucţiunea.

Instrucţiunile JPE şi JPO (salt dacă paritatea este impară) sînt special folosite pentru testarea parităţii datelor de intrare. Oricum instrucţiunea IN nu setează flagurile. Acestea pot fi setate prin adunarea lui OOH la conţinutul acumulatorului.

EAH =

1 1 1 0 1 0 1 0
baitul inferior adresă
baitul superior adresă

JPE addr

Cicluri : 3 (2 sau 3 la 8085)
Stări : 10 (7 sau 10 la 8085)
Adresare: imediată
Flaguri afectate : nu

JPO **Salt dacă paritatea este impară**

Paritatea este impară dacă baitul din acumulator conține un număr impar de biți 1. Condiția aceasta este indicată prin Z = O.

Instrucțiunea JPO testează indicatorul P. Dacă numărul de biți 1 este impar (P=0) programul execută un salt la adresa specificată în instrucțiunea JPO. Dacă numărul de biți 1 este par (P=1) programul continuă cu executarea următoarei instrucțiuni secvențiale.

Codul de bperare	Operandul
JPO	adresa

Adresa poate fi specificată ca un număr, o etichetă sau o expresie. Asamblorul inversează baiții (superior și inferior) ai adresei, cînd asamblează instrucțiunea.

Instrucțiunile JPO și JPE (salt dacă paritatea este pară) sînt special folosite pentru testarea parității datelor de intrare. Oricum instrucțiunea IN nu setează nici un indicator de condiție (flag). Indicatorii pot fi setați prin adunarea lui 00H la conținutul acumulatorului.

E2H =	1 1 1 0 0 0 1 0	JP addr
	baitul inferior adresă	
	baitul superior adresă	

Cicluri : 3 (2 sau 3 la 8085)
Stări : 10 (7 sau 10 la 8085)
Adresare: imediată
Flaguri afectate: nici unu

Exemplu :

Acest exemplu arată trei metode diferite dar echivalente pentru saltul la unul din două puncte ale programului, în funcție de bitul de semn S al numărului.

Se presupune că baitul care trebuie să fie testat este registrul C.

Ericheta	Codul	Operandul
ONE:	MOV	A,C,
	ANI	80H
	JZ	PLUS
	JNZ	MINUS
TWO:	MOV	A,C.
	RLC	
	JNC	PLUS
	JMP	MINUS
THREE:	MOV	A,C
	ADI	0
	JM	MINUS
PLUS:	-	; Bit de semn = 0
MINUS:	-	; Bit de semn = 1

Instrucţiunea directă ANI din blocul cu etocheta ONE trece în zero toţi biţii baitului de date cu excepţua bitului S (de semn) care rămîne neschimbat. Dacă bitul de semn este zero bitul de condiţie Z va fi 1 şi instrucţiunea JZ transferă con - trolul programului la instrucţiunea de la PLUS. In caz contrar, cînd Z = 0, instrucţiunea JZ va modifica cu 3 cøntorul de pro - gram şi în continuare se va executa instrucţiunea JNZ care va transfera controlul programului la instrucţiunea de la MINUS. (Bitul Z nu esre afectat de nici-o instrucţiune JUMP).

Instrucţiunea RLC din blocul cu eticheta TWO va face ca bitul CY să fie făcut egal cu bitul S al bitului de date.

Dacă bitul S (de semn) este 0, instrucţiunea JNC provoacă un salt la PLUS. In caz contrar, este executată instrucţiunea JMP care va transfera necondiţionat controlul programului la MI- NUS. (De notat că, în acest moment, o instrucţiune JC poate fi substituită cu un JUMP necondiţionat cu rezultate identice).

Instrucţiunea directă de adunare ADI din blocul cu eticheta THREE are infleunţă asupra biţilor de condiţie, Dacă bitul de semn a fost setat (S=1) instrucţiunea JM transferă controlul programului la MINUS. In caz contrar, controlul programului de - curge automat cu rutina PLUS.

JZ — Salt dacă este zero

Instrucţiunea JZ testează modificarea bitului Z (de zero). Dacă Z = 1, executarea programului este transferată la adresa specificată în instrucţiunea JZ. Dacă Z = 0, programul continuă cu executarea următoarei instrucţiuni secvenţiale.

Codul de operare	Operandul
JZ	adresa

Adresa poate fi specificată ca un număr, o etichetă sau o expresie. Asamblorul inversează baiţii (superior şi inferior) ai adresei cînd asamblează instrucţiunea.

CAH =		JZ addr
	1 1 0 0 1 0 1 0	
	baitul inferior adresă	
	baitul superior adresă	

Cicluri : 3 (2 sau 3 la 8085)
Stări : 10 (7 sau 10 la 8085)
Adresare : imediată
Flaguri afectate : nu

LDA — Incarcă acumulatorul direct

Instrucţiunea LDA încarcă acumulatorul cu o copie a baitului din memorie ce se găseşte în locaţoa specifică de baiţii doi şi trei (superior şi inferior) ai instrucţiunii LDA.

Codul de operare	Operandul
LDA	adresa

Adresa poate fi exprimată ca un număr. o etichetă definită anterior sau o expresie. Asamblorul inversează baiţii (superior şi inferior) adresei cînd construieşte instrucţiunea.

3AH =		LDA addr
	0 0 1 1 1 0 1 0	
	baitul inferior adresă	
	baitul superior adresă	

Cicluri : 4
Stări : 13
Adresare: directăr
Flaguri afectate : nu

Exemplu :

Următoarele instrucţiuni sînt echivalente. Cînd sînt executate, fiecare înlocuieşte conţinutul acumulatorului cu baitul de date stocat în locaţia 300H din memorie.

LOAD : LDA 300H
LDA 3*(16*16)
LDA 200H + 256

LDAX Incarcă acumulatorul indirect

Instrucţiunea LDAX încarcă acumulatorul cu o copie a baitului stocat în locaţia din memorie, adresată de registrul pereche B, sau de registrul pereche D.

Codul de operare	Operandul
LDAX	{B, D}

Operandul B specifică registrele pereche B şi C, iar D specifică registrele pereche D şi E. Această instrucţiune nu poate specifica decît registrele pereche B sau D.

OAH sau 1AH = | 0 0 0 | r | 1 0 1 0 | LDAX B sau LDAX D

0 = registrul pereche B
1 = registrul pereche D

Cicluri : 2
Stări : 7
Adresare : registru indirect
Flaguri afectate : nu

Exemplu :

Se presupune că registrul D conţine valoarea 93H şi registrul E conţine 8BH. Următoarea instrucţiune încarcă asumulatorul cu conţinutul locaţiei din memorie 938BH :

LDAX D

LHLD Incarcă H şi L direct

Instrucţiunea LHLD încarcă registrul L cu o copie a baitului stocat la locaţia din memorie specificată de baiţii doi şi trei (superior şi inferior) ai instrucţiuni LHLD. LHLD încarcă registrul H cu o copie a baitului stocat la următoarea locaţie superioară din memorie.

Codul de operare	Operandul
LHLD	adresa

Adresa poate fi formată dintr-un număr, o etichetă sau o expresie.

Anumite instrucţiuni folosesc simbolul de referinţă M pentru accesul la locaţia din memorie curent specificată de registrele H şi L

Instrucţiunea LHLD este una din instrucţiunile prevăzute pentru încărcarea unei noi adrese în registrele H şi L.

Utilizatorul poate de asemenea să încarce partea superioară a stivei în registrele H şi L (instrucţiunea POP de extragere în stivă).

Amb le instrucţiuni LHLD şi POP înlocuieşte conţinutul registrelor H şi L.

Se poate de asemenea schimba conţinutul registrelor H şi L cu registrele D şi E (instrucţiunea XCHG) sau cu partea superioară a stivei (instrucţiubea XTHL), dacă se doreşte să se păstreze registrele H şi L pentru a fi folosite ulterior.

Instrucţiunea SHLD stochează pe H şi L în memorie.

2AH =	0 0 1 0 1 0 1 0	LHLD addr
	baitul inferior adresă	
	baitul superior adresă	

Cicluri : 5
Atări : 16
Adresare: directă
Flaguri afectate : nu

Exemplu :

Se presupune că locaţiile 3000 H şi 3001 H conţine adresa 064EH stocată sub forma 4E06. In următoarea secvenţă de pro-

gram, instrucţiunea MOV mută o copie a baitului stocat la adresa 064E în acumulator.

```
LHLD  3000 H      ; Incarcă adresa
MOV   A,M         ; Incarcă acumulatorul de la adresă
```

LXI **Incarcă registrul pereche imediat**

Instrucţiunea LXI este formată din trei baiţi; cel de al doilea şi al treilea bait conţin datele sursă care trebuie să fie încărcate în registrul pereche.

Instrucţiunea LXI încarcă un registru pereche copiind cel de al doilea şi al treilea bait al său în registru pereche destinaţie.

Codul de operare	Operandul
LXI	{B, D, H, SP} , date

Primul operand trebuie să specifice registrul pereche ce trebuie încărcat.

Instrucţiunea LXI poate încărca registrele pereche B şi C, D şi E, H şi L şi registrul indicator de stivă SP.

Al doilea operand specifică cei doi baiţi de date ce vor fi încărcaţi. Aceste date pot fi codate sub forma unui număr, unei constante ASCII, eticheta unor valori definite anterior sau o expresie.

Datele nu trebuie să depăşească doi biţi.

Instrucţiunea LXI este singura instrucţiune imediată care acceptă valori de 16 biţi. Toate celelalte instrucţiuni imediate cer valori de 8 biţi.

De notat că asamblorul inversează cei doi baiţi de date pentru a obţine formatul adresei stocate în memorie.

Instrucţiunea LXI încarcă cel de al treilea bait al său în primul dintre registrele pereche şi cel de al doilea bait al său în al doilea dintre registrele pereche. Aceasta are ca efect reinversarea datelor obţinîndu-se formatul necesar adresei stocate în registre.

Astfel instrucţiunea LXI B,'AZ' încarcă pe A în registrul B şi pe Z în registrul C.

01H, 11H, 21H, 31H =

0 0	R P	0 0 0 1
baitul inferior date		
baitul superior date		

RP =
- 00 pentru LXI B
- 01 pentru LXI D
- 10 pentru LXI H
- 11 pentru LXI SP

Cicluri : 3
Stări : 10
Adresare : imediat
Flaguri afectate : nu

Exemplu :

O utilizare frecventă a instrucţiunei LXI este stabilirea adresei din memorie pentru folosirea următoarei instrucţiuni. In următoarea secvenţă de program, instrucţiunea LXI încarcă adresa lui STRNG în registrele H şi L. Instrucţiunea MOV încarcă apoi în acumulator datele stocate la această adresă.

```
LXI  H,STRNG        ; SET ADDRESS
MOV  A,M            ; LOAD STRNG INTO ACCUMULATOR
```

Următoarea instrucţiune LXI este folosită pentru a iniţializa pointerul de stivă la un modul relocabil.

Programul LOCATE prevede o adresă pentru eticheta STACK special rezervată.

```
LXI SP,STACK
```

<u>MOV</u> <u>Deplasează (transferă)</u>

Instrucţiunea MOV deplasează un bait de date copiind cîmpul sursă în cîmpul destinaţie. Datele sursă rămîn neschimbate . Operanzii instrucţiunii specifică dacă deplasarea se face de la un registru la un alt registru, de la un registru la memorie sau de la memorie la un registru.

<u>Transferă registru la registru</u>

Codul de operare	Operandul
MOV	reg 1, reg 2

Instrucţiunea copiază conţinutul registrului 2 în registrul 1. Fiecare operand trebuie să specifice unul din registrele A,B,C,D,E,H sau L.

Cînd acelaşi registru este specificat de ambii operanzi (de exemplu MOV A,A) instrucţiunea funcţionează ca NOP (No Operaţion).

40H ... 47H
48H ... 4FH
50H ... 57H
58H ... 5FH
60H ... 67H
68H ... 6FH
78H ... 7FH

=

0 1	D D D	S S S

Cicluri : 1
Stări : 5(4 la 8085)
Adresare: registru
Flaguri afectate: nu

MOV B, reg
MOV C, reg
MOV D, reg
MOV E, reg
MOV H, reg
MOV L, reg
MOV A, reg

Transferă în memorie

Codul de operare	Operandul
MOV	M, reg

Această instrucţiune copiază conţinutul registrului specificat în locaţia din memorie ce a fost adresată de registrul H şi L.

M se referă la baitul adresat de registrul pereche H şi L.

Al doilea operand trebuie să numească unul din registre. Instrucţiunea MOV M,M nu este permisă.

70H ... 75H şi 77H =

0 1 1 1 0	S S S

MOV M,reg

Cicluri : 2
Stări : 7
Adresare: registru indirect
Flaguri afectate : nu

Transferă din memorie

Codul de operare	Operandul
MOV	reg,M

Această instrucţiune copiază conţinutul unei locaţii din memorie adresată de registrul L şi H, în registrul specificat de instrucţiune. Cel de al doilea operand trebuie să fie M.

M este un simbol de referire la registrele H şi L.

Cod		Format			Instrucţiune
46H	=	0 1	D D D	1 1 0	MOV B,M
4EH					MOV C,M
56H					MOV D,M
5EH					MOV E,M
66H					MOV H,M
6EH					MOV L,M
7EH					MOV A,M

Cicluri : 2
Stări : 7
Adresare: registru direct
Condiţii flag : nu

Exemple :

Eticheta	Cod de operare	Operand	Comentariu
LDACC :	MOV	A,M	; Incarcă acumulatorul din memorie
	MOV	E,A	; Copiază acumulatorul în reg.E
NULOP :	MOV	C,C	; NULL OPERATION

MVI — Transferă imediat

Instrucţiunea MVI are doi baiţi ; cel de al doilea bait conţine datele ce trebuie să fie transferate .

Instrucţiunea MVI transferă un bait de date copiind cel de al doilea bait al său în cîmpul destinaţie.

Operanzii instrucţiunii specifică destinaţia transferului, către un registru sau memorie.

Transfer direct către registru

Codul de operare	Operandul
MVI	reg,date

Primul operand "reg" trebuie să numească unul din registrele A la E, H sau L ca destinaţie a transferului.

Cel de al doilea operand "date" specifică datele reale ce vor di transferate. Aceste date pot fi de forma unui număr, o constantă ASCII, o etichetă a cărei valoare a fost anterior definită, sau o expresie.

Datele nu trebuie să depăşească un bait.

Asamblorul se caracterizează prin tratarea tuturor simbolurilor externe şi relocabile ca adresa de 16 biţi. Cînd unul din aceste simboluri apare în expresia operandului instrucţiunii directe, el trebuie să fie precedar de unul din operatorii HIGH sau

LOW care să precizeze care din baiţii adresei trebuie să fie folosit în evaluarea expresiei.

Cînd lipseşte operatorul, asamblorul consideră un operator LOW şi dă un mesaj de eroare.

Cod		8		Instrucţiune
06H	=	0 0 | D D D | 1 1 0		MVI B, date
0EH				MVI C, date
16H		Cicluri : 2		MVI D, date
1EH		Stări : 7		MVI E, date
26H		Adresare : imediată		MVI H, date
2EH		Flaguri afectate : nu		MVI L, date
3EH				MVI A, date

<u>Transferă direct către memorie</u>

Codul de operare	Operandul
MVI	M, date

Această instrucţiune copiază datele stocate în cel de al doilea bait al său, în locaţia din memorie adresată de registrul H şi L. M este un simbol de referire la registrul pereche H şi L.

36H	=	0 0 1 1 0 1 1 0	MVI M, date
		date	

Cicluri : 3 3
Stări : 10
Adresare: direct/registru indicat
Flaguri afectate : nu

Exemple :

Următoarele exemple arată diverse metode de folosire a instrucţiunii MVI, pentru trabsfer direct de date. Toate exemplele generează grupul de biţi pentru caracterul A din codul ASCII. (01ooooo1B = 41H = 65D= ... = caracterul A') .

```
MVI M, 01000001B
MVI,M,'A'
MVI M.41 H
MVI M, 101Q
MVI M,65
MVI M,5 + 30*2
```

NOP Inoperant

Instrucţiunea NOP nu îndeplineşte nici-o operaţie şi nu are efect asupra flagurilor de condiţie.

Instrucţiunea NOP este folosită pentru completarea unor bucle de timp.

Codul de operare	Operandul
NOP	

La instalaţia NOP nu este permis operand.

ORA SAU inclusiv cu aculuiatorul

Instrucţiunea ORA îndeplineşte o operaţiune logică SAU INCLUSIV între conţinutul baitului specificat şi conţinutul acumulatorului. Rezultatul este plasat în acumulator.

Sumar al operaţiunilor logice

Operaţiunea SI produce un bit "1" ca rezultat, numai cînd biţii corespunzători din datele testate şi mască sînt 1.

Operaţiunea SAU produce un bit "1" ca rezultat cînd biţii corespunzători în datele de test sau mască sînt 1.

Operaţiunea SAU EXCLUSIV produce un bit "1" ca rezultat numai cînd biţii corespunzători în datele de test şi datele mască sînt diferiţi.

SI	SAU	SAU EXCLUSIV
10101010	10101010	10101010
00001111	00001111	00001111
00001010	10101111	10100101

SAU între registru şi acumulator

Codul de operare	Operandul
ORA	reg

Operandul trebuie să specifice unul din registrele A la E, H sau L. Această instrucţiune execută operaţiunea logică SAU între conţinutul tegistrului indicat de operand şi acumulator apoi stochează rezultatul în acumulator. Flagurile CY şi AC sînt trecute în zero.

B0H .. B5H şi B7H = | 1 0 1 1 0 | S S S | ORA reg

reg = B,C,D,
E,H,L,A

Cicluri : 1

Stări : 4

Adresare : registru

Flaguri afectate: Z,S,P,CY,AC

SAU între memorie şi acumulator

Codul de operare	Operandul
ORA	M

Operaţiunea SAU inclusiv se execută între conţinutul locaţiei din memorie specificată de registrul pereche H şi L şi conţinutul acumulatorului, Rezultatul este stocat în acumulator. Flagurile CY şi AC sînt trecute în zero.

B6H = | 1 0 1 1 0 1 1 0 | ORA M

Cicluri : 2

Stări : 7

Adresare: registru indirect

Flaguri: Z,S.P.CY,AC

Deorece în operaţiunea SAU dintre oricare bit cu un bit de valoare 1 se obţine un bit 1 şi dintre oricare bit cu un bit de valoare zero, bitul rămîne neschimbat, instrucţiunea ORA se foloseşte frecvent pentru trece în 1 un anumit bit sau o grupă de biţi.

In exemplul următor se trece în 1 bitul 3 restul biţilor rămînînd neschimbaţi. Aceasta se face frecvent cînd biţi individuali sînt folosiţi în program ca indicatori de stare (flaguri).

Se presupune că registrul D conţine valoarea 08H.

Număr bit	7 6 5 4 3 2 1 0
Acumulatorul	0 1 0 0 0 0 1 1
Registrul D	0 0 0 0 1 0 0 0
	0 1 0 0 1 0 1 1

ORI — SAU inclusiv imediat

Instrucţiunile ORI execută o operaţiune logică SAU inclusiv între conţinutul celui de al doilea bait al instrucţiunii şi conţinutul acumulatorului. Rezultatul este plasat în acumulator. Instrucţiunea ORI, de asemenea trece în zero flagurile CY şi AC.

Codul de operare	Operandul
ORI	data

Operandul trebuie să specifice datele ce vor fi folosite în operaţiunea SAU. Aceste date pot fi forma unui număr, o constantă ASCII, o etichetă a cărei valoare a fost anterior definită, sau p expresie. Datele nu pot depăşi un bait.

Asamblorul se caracterizează prin tratarea tuturor simbolurilor externe şi relocatabilă ca adrese de 16 biţi.

Cînd unul din aceste simboluri apare în expresia operandului, el trebuie precedat de operatorii HIGH sau LOW care să precizeze care din baiţii adresei trebuie să fie folosit în evaluarea expresiei.

Cînd lipseşte operatorul, asamblorul consideră un operator LOW şi dă un mesaj de eroare.

F6H =

1	1	1	1	0	1	1	0
date							

ORI date

Cicluri : 2
Stări : 7
Adresare: imediată
Flaguri afectate : Z,S,P,CY,AC

Exemple :

Următoarele exemple prezintă o serie de metode pentru definirea instrucţiunii ORI cu date directe.

Toate exemplele generează secvenţa de biţi pentru caracterul A din codul ASCII (01000001B = 41 H = 65 D = ... = caracterul 'A').

```
ORI 01000001B
ORI 'A'
ORI 41H
ORI 101Q
ORI 65
ORI 5+30*2
```

OUT **Ieşire către periferice**

Instrucţiunea OUT transferă pe busul de date de 8 biţi datele (un bait = 8 biţi) conţinute în acumulator şi numărul portului selectat pe busul de adrese de 16 biţi.

Detalii asupra funcţiunilor intrare/ieşire pot fi obţinute din manualul : "8080 or 8085 Microcomputer System User's Manual".

Codul de operare	Operandul
OUT	exp

Operandul trebuie să specifice numărul perifericului (portului) dorit. Acesta poate fi de forma unui număr sau a unei expresii în domeniul 00h la OFFH.

D3H =	1	1	0	1	0	0	1	1	OUT exp
	exp								

Cicluri : 3
Stări : 10
Adresare : direct
Flaguri afectate : nu

PCHL **Transferă H şi L în contorul programului**

Instrucţiunea PCHL încarcă conţinutul registrului H şi L în registrul PC (Program Counter). Deoarece procesorul extrage următoarea instrucţiune de la adresa stabilită de PC, instrucţiunea PCHL are efectul unei instrucţiuni JUMP.

Codul de operare	Operandul
PCHL	

Operandul nu este admis în instrucţiunea PCHL.

Instrucţiunea PCHL transferă conţinutul registrului H

în baitul superior al PC şi conţinutul registrului L în baitul inferior al PC.

E9H = 1 1 1 0 1 0 0 1   PCHL

Cicluri : 1
Stări : 5 (6 la 8085)
Adresare: registru
Flaguri afectate : nu

Exemplu :

Una din tehnicile de trecere a datelor la o subrutină este de a plasa datele imediat după chemarea subrutinei (CALL).

Adresa de revenire introdusă în stivă de instrucţiubea CALL, de fapt adresează datele mai înainte de următoarea ins - trucţiune, după CALL. In acest exemplu se presupune că doi baiţi de date urmează după CALL. Următoarea secvenţă codată provoacă o revenire la următoarea instrucţiune, după CALL.

```
GOBACK : POP H   ; In adresa datelor
         INR L   ; Adună 2 la formatul
         INR L   ; adresei de intoarcere
         PCHL    : RETURN ( întoarcere)
```

<u>POP</u>   <u>Extragere din stivă</u>

Instrucţiunea POP extrage doi baiţi de date din stivă şi îi copiază într-un registru pereche sau copiază conţinutul registrului PSW (Processor's Status Word) în Acumulator.

<u>Instrucţiunea POP registru pereche</u>

Această instrucţiune POP copiată conţinutul locaţiei din memorie adresată de pointerul de stivă SP, în registrul inferior al perechii de registre. Apoi instrucţiunea POP incrementează cu 1 pointerul de stivă şi copiază conţinutul de la adresa rezultată în registrul superior al perechii de registre.

După acestea instrucţiunea POP incrementează din nou registrul SP pentru a indica următoarea valoare mai veche din stivă.

Codul de operare	Operandul
POP	{B, D, H, PSW}

Operandul poate specifica unul din registrele pereche B şi C, D şi E sau H şi L. Instrucţiunea POP este explicată separat.

C1H, D1H, E1H = | 1 1 | R P | 0 0 0 1 | — POP B, POP D, POP H

Cicluri : 3
Stări : 10
Adresare: registru intrare
Flaguri afectate : nu

Instrucţiunea POP PSW

Instrucţiunea POP PSW foloseşte conţinutul locaţiei din memorie, specificată de pointerul de stivă SP, pentru a restabili indicatorii de condiţie.

Instrucţiunea POP PSW incrementează apoi registrul SP cu 1 şi copiază conţinutul de la această adresă în acumulator. După aceastea instrucţiunea POP incrementează din nou registrul SP pentru a indica următoarea valoare mai veche din stivă.

F1H = | 1 1 1 1 0 0 0 1 | POP PSW

Cicluri : 3
Stări : 10
Adresare: registru indirect
Flaguri afectate : Z,S,P,CY,AC

Exemplu :

Se presupune că este apelată o subrutină datorită unei intreruperi externe. In general orice subrutină trebuie să salveze şi să restabilească conţinutul oricărui registru folosit astfel ca programul de bază să poată continua normal, cînd reia controlul, după ce întreruperea a fost deservită.

Următoarea secvenţă de instrucţiuni PUSH şi POP salvează şi restabileşte conţinutul tuturor registrelor şi al PSW.

```
PUSH PSW
PUSH B
PUSH D
PUSH H
  .
  .
SUBRUTINA
  .
  .
POP H
POP D
POP B
POP PSW
RET
```

De notat că sexvența de instrucțiuni POP, de extragere din stivă, este inversă secvenței de instruțiuni PUSH, de introducere în stivă, dacă stiva este de tipul LIFO (ultimul la intrare, primul la ieșire).

<u>PUSH</u> <u>Introducere în stivă</u>

Instrucțiunea PUSH copiază doi biți de date în stivă. Aceste date pot fi : conținutul unor registre pereche sau al registrului PSW după cum se va putea vedea în continuare.

<u>PUSH registru pereche</u>

Instrucțiunea PUSH decrementează pointerul de stivă SP cu 1 și copiază conținutul registrului superior al perechii de registre la adresa rezultată din SP. Instrucțiunea PUSH decrementează apoi din nou pointerul de stivă și copiază conținutul registrului inferior al perechii de registre la adresa rezultată din SP.

Registrele sursă rămîn neschimbate.

<u>Codul de operare</u>	<u>Operandul</u>
PUSH	{ B, D, H, PSW }

Operandul poate să specifice unul din registrele pereche B și C, D și E sau H și L. Instrucțiunea PUSH PSW este explicată separat.

C5H D5H E5H	=	1 1 Rp 0 1 0 1	PUSH B PUSH D PUSH H

Cicluri : 3
Stări : 10(13 la 8085)
Adresare: registru indirect
Flaguri afectate : nu

Exemplu :

Se presupune că registrul B conţine 2AH, registrul C conţine 4CH şi registrul pointer de stivă SP conţine 9AAFH.

Instrucţiunea PUSH B stochează 2AH (conţinutul registrului B) la adresa din memorie 9AAFH - 1 = 9AAEH şi pe 4 CH (conţinutul registrului C) la adresa din memorie 9AAEH-1 = 9AADH.

Pointerul de stivă SP este modificat la 9AADH :

	Stiva înainte, de PUSH	Adresa	Stiva după PUSH	
SP înainte	XX	9AAF	XX	
	XX	9AAE	2A	
	XX	9AAD	4C	SP după
	XX	9AAC	XX	

PUSH PSW

Instrucţiunea PUSH PSW copiază conţinutul registrului PSW în stivă Registrul de stare al programului PSW cuprinde conţinutul acumulatorului indicatorii de condiţie curenţi.

Deoarece indicatorii de condiţie sînt în număr de 5 aceştia sînt incluşi într-un bait (8 biţi) în felul următor:

7	6	5	4	3	2	1	0
S	Z	0	AC	0	P	1	CY

La microprocesorul 8080 biţii 3 şi 5 sînt întodeauna zero iar bitul 1 este întodeauna 1. La microprocesorul 8085 aceşti biţi nu sînt definiţi.

Instrucţiunea PUSH PSW decrementează cu 1 pointerul de stivă SP şi copiază conţinutul acumulatorului la adresa indicată de SP. După aceasta instrucţiunea PUSH PSW decrementează din nou cu 1 pointerul de stivă şi copiază octetul indicatorilor de condiţie la adresa dată de SP. Conţinutul acumulatorului şi indicatorii de condiţie rămîn neschimbate.

F5H = | 1 1 1 1 0 1 0 1 | PUSH PSW

Cicluri : 3
Stări : 11 (12 la 8085)
Adresare: registru indir
Flaguri afectate : nu

Exemplu

Cînd un program apelează o subrutină, întodeauna este necesar să se conserve stările programului curent astfel ca atunci cînd acesta va prelua din nou controlul să se poată desfăşura normal în continuare din punctul şi în condiţiile de stare avute la întrerupere. Normal subrutina îndeplineşte o instrucţiune PUSH PSW înainte de executarea oricărei alte instrucţiuni care ar putea altera conţinutul acumulatorului şi modifica condiţiile flag.

Subrutina apoi restabileşte starea anterioară a sistemului prin executarea unei instrucţiuni POP PSW înainte de a se reda controlul programului de bază.

RAL Rotire stînga prin CY

Instrucţiunea RAL roteşte cu un bit spre stînga conţinutul acumulatorului şi a bitului CY.

Bitul de transport CY este tratat ca şi cum ar face parte din acumulator. Astfel, bitul CY este trecut în locul bitului din acumulator cu ponderea cea mai mică,conţinutul acumulatorului este deplasat spre stînga cu un bit,iar bitul cu ponderea cea mai mare rezultat este trecut în locul bitului CY.

Codul de operare	Operandul
RAL	

Nici-un operand nu este permis în instrucţiunea RAL.

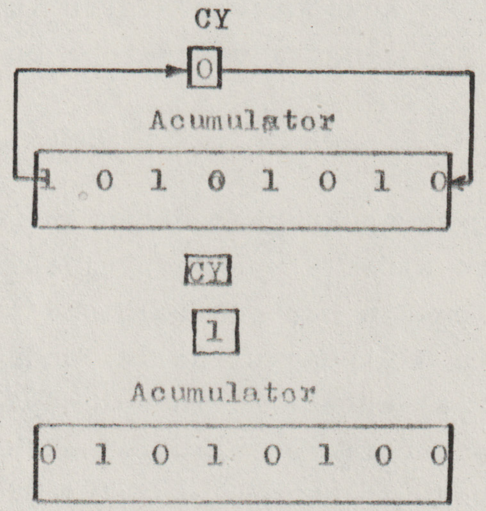
17H = | 0 0 0 1 0 1 1 1 | RAL

Cicluri : 1
Stări : 4
Flaguri afectate : numai CY

Exemplu :

Se presupune că acumulatorul conţine valoarea OAAH iar CY=0. In următoarea diagramă este ilustrat efectul pe care îl are instrucţiunea RAL asupra acumulatorului.

înainte



După instrucţiunea RAL

<u>RAR</u>   <u>Rotire dreapta prin CY</u>

Instrucţiunea RAR, roteşte cu un bit spre dreapta conţinutul acumulatorului şi a bitului CY.

Bitul CY este tratat ca şi cum ar face parte din acumulator. Astfel, bitul CY este trecut în locul bitului din acumulator cu ponderea cea mai mare, conţinutul acumulatorului este deplasat spre dreapta cu un biţ. iar bitul cu ponderea cea mai mică rezultat din deplasare este trecut în locul bitului CY.

<u>Codul de operare</u>	<u>Operandul</u>
RAR	

Nici-un operand nu este permis în instrucţiunea RAR.

1FH = | 0 0 0 1 1 1 1 1 | RAR

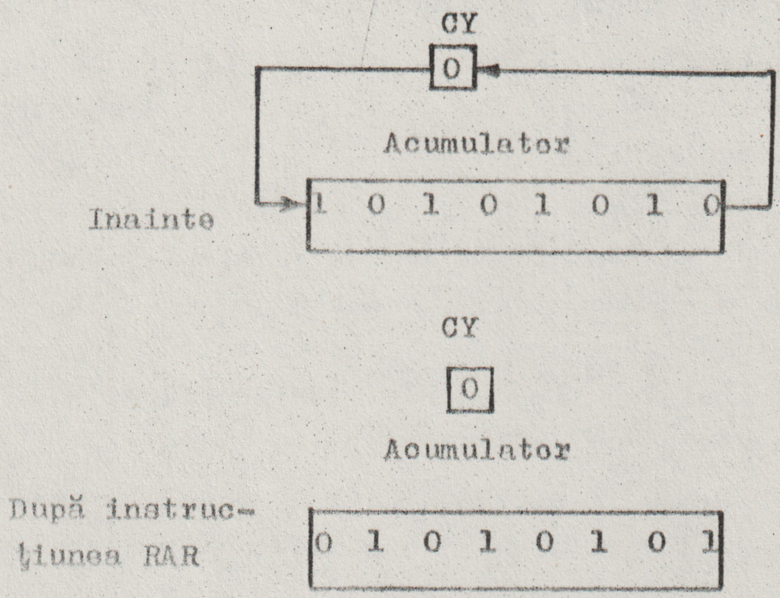
Cicluri : 1

Stări : 4

Flaguri afectate: numai CY

Exemplu :

Se presupune că acumulatorul conţine valoarea OAAH iar CY=0. In următoarea diagramă este ilustrat efectul pe care îl are instrucţiunea RAR asupra acumulatorului.



<u>RC</u> <u>Retur dacă există transport CY = 1</u>

Instrucţiunea RC testează condiţia flag CY. Dacă CY = 1 pentru a indica un transport instrucţiunea extrage doi biţi din stivă şi îi plasează în contorul programului PC.

Execuţia programului presupune continuarea la noua adresă din contorul programului.

Dacă CY = 0 execuţia programului continuă cu următoarea instrucţiune secvenţială.

<u>Codul de operare</u>	<u>Operandul</u>
RC	

In instrucţiunea RC nu este permis nici-un operand.

D8H = | 1 1 0 1 1 0 0 0 | RC

Cicluri : 1 sau 3
Stări : 5 sau 11 (6 sau 12 la 8085)
Adresare: registru indirect
Flaguri afectate: nu

RET Retur de la subrutină

Instrucţiunea RET extrage doi baiţi din stivă şi îi plasează în registrul PC. Execuţia programului continuă la noua adresă indicată de contorul programului.

Normal, instrucţiunea RET este folosită împreună cu instrucţiunea CALL. (Această condi-ie se aplică la toate variantele acestor instrucţiuni).

In acest caz s-a presupus că datele extrase din stivă de instrucţiunea RET reprezintă adresa de retur plasată aici de instrucţiunea CALL precedentă. Acestea au ca efect preluarea controlului de către următoarea instrucţiune după CALL. Utilizatorul trebuie să fie sigur că instrucţiunea RET a extras din stivă o adresă cu cod executabil.

Dacă instrucţiunea a extras o adresă cu date procesorul încearcă să execute datele ca şi cum acestea ar reprezenta un cod.

Codul de operare	Operandul
RET	

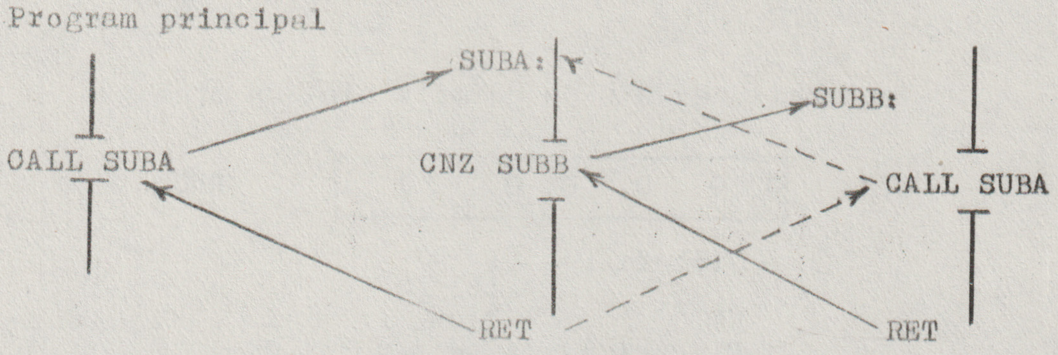
Instrucţiunea RET nu primeşte nici-un operand.

C9H = | 1 1 0 0 1 0 0 1 | RET

Cicluri : 3
Stări : 10
Adresare: registru indirect
Flaguri afectate: nu

Exemplu

In următoarea diagramă este dat un exemplu de folosire a instrucţiunii RET.



RIM (numai pentru procesorul 8085)

Instrucţiunea RIM încarcă acumulatorul cu următoarele informaţii:

7	6	5	4	3	2	1	0
SID	17	16	15	IE	7.5	6.5	5.5

- biţii 2–0 (7.5, 6.5, 5.5): Măşti de întrerupere 1=mască
- bitul 3 (IE): Indicator de permisie a întreruperii 1=permis
- biţii 6–4 (17, 16, 15): Intreruperi indecise 1=nehotărît,nedecis
- bitul 7 (SID): Bit de date al intrării seriale,dacă există

Indicatorul de condiţie IE se referă la întregul sistem de întreruperi. Astfel indicatorul IE este identic ca funcţie şi nivel cu primul INTE al lui 8080

IE = 1 indică permiterea întreruperilor.

20H = | 0 0 1 0 0 0 0 0 | RIM

Cicluri : 1

Stări : 4

Flaguri afectate: nu

RLC Rotire stînga acumulator

Instrucţiunea RLC modifică bitul CY făcîndu-l egal cu bitul de pondere superioară din acumulator. Instrucţiunea roteşte spre stînga cu un bit acumulatorul şi transferă bitul din poziţia cu pondere superioară,rezultat din rotire,în poziţia cu ponderea cea mai mică a acumulatorului.

Codul de operare	Operandul
RLC	

Instrucţiunea RLC nu permite nici-un operand.

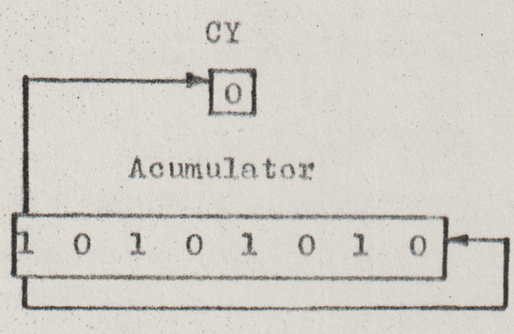
07H | 0 0 0 0 0 1 1 1 | RLC

Cicluri : 1
Stări : 4
Flaguri afectate : numai CY

Exemplu :

Se presupune că acumulatorul conţine valoarea OAAH şi CY = 0. In următoarea diagramă este ilustrat efectul pe care îl are instrucţiunea RLC asupra acumulatorului.

Inainte



CY

[1]

Acumulator

După instruc-
ţiunea RLC

0 1 0 1 0 1 0 1

Cicluri : 1 sau 3
Stări : 5 sau 11 (6 sau 12 la 8085)
Adresare: registru indirect
Flaguri afectate: nu

RM Retur dacă semnul este minus

Instrucţiunea RM testează condiţia de semn S.Dacă S=1 indicînd că datele din acumulator sînt negative,instrucţiunea extrage doi baiţi din stivă şi îi plasează în registrul contorului de program PC. Programul continuă execuţia cu noua adresă din PC. Dacă S = 0 programul execută următoarea instrucţiune secvenţială .

Codul de operare	Operandul
RM	

Instrucţiunea RM nu permite nici-un operand.

F8H = | 1 1 1 1 | 1 0 0 0 | RM

Cicluri : 1 sau 3
Stări : 5 sau 11 (6 sau 12 la 8085)
Adresare: registru indirect
Flaguri afectate : nu

RNC Retur dacă nu este transport

Instrucţiunea RNC testează condiţia CY. Dacă CY=0 indicînd prin aceasta că nu este un transport, instrucţiunea RNC extrage doi baiţi din stivă şi îi plasează în registrul contorului de program PC. Programul continuă execuţia cu noua adresă din PC. Dacă CY = 1 programul execută următoarea instrucţiune secvenţială.

Codul de operare	Operandul
RNC	

Nici-un operand nu este permis în instrucţiunea RNC.

DOH = | 1 1 0 1 0 0 0 0 | RNC

Cicluri : 1 sau 3
Stări : 5 sau 11 (sau 12 la 8085)
Adresare : registru indirect
Flaguri afectate : nu

RNZ Retur dacă nu este zero

Instrucţiunea RNZ testează condiţia Z. Dacă Z = 0 indicînd că conţinutul acumulatorului este diferit de zero, instrucţiunea RNZ extrage doi biţi din stivă şi îi plasează în registrul contorului de program PC. Programul continuă execuţia cu noua adresă din PC. Dacă Z = 1 programul execută următoarea instrucţiune secvenţială .

Codul de operare | Operandul
RNZ

Nici-un operand nu este permis în instrucţiunea RNZ.

COH = | 1 1 0 0 0 0 0 0 | RNZ

Cicluri : 1 sau 3
Stări : 5 sau 11 (6 sau 12 la 8085)
Adresare: registru indirect
Flaguri afectate : nu

RP Retur dacă semnul este plus

Instrucţiunea RP testează condiţia de semn S. Dacă S=0, indicînd că datele din acumulator sînt pozitive, instrucţiunea extrage doi baiţi din stivă şi îi plasează în registrul de program PC. Programul continuă execuţia cu noua adresă din PC. Dacă S = 1 programul execută următoarea instrucţiune la rînd.

Codul de operare | Operandul
RP

Nici-un operand nu este permis în instrucţiunea RP.

FOH = | 1 1 1 1 0 0 0 0 | RP

Cicluri : 1 sau 3
Stări : 5 sau 11 (6 sau 12 la 8085)
Adresare: registru indirect
Flaguri afectate : nu

RPE Retur dacă paritatea este pară

Paritatea este pară dacă baitul din acumulator conţine un număr par de biţi de valoare 1. Această stare este indicată prin dondiţia P = 1.

Instrucţiunea RPE şi RPO sînt folosite pentru testarea parităţii datelor de intrare. Printre altele, o instrucqiune IN nu trebuie să modifice indicatorii de condiţie.

Pentru a se evita alterarea datelor, indicatorii de condiţie pot fi modificaţe prin adunarea valorii 00H la conţinutul acumulatorului.

Instrucţiunea RPE testează condiţia de paritate P. Dacă P = 1 indiqîndu-se astfel o paritate pară, instrucţiunea RPE extrage doi baiţi din stivă şi îi plăsează în registrul contorului de program PC. Programul continuă execuţia cu noua adresă din PC. Dacă P = 0 programul execută următoarea instrucţiune secvenţială.

Codul de operare	Operandul
RPE	

Nici un operand nu este admis în instrucţiunea RPE.

E8H =	1	1	1	0	1	0	0	0		RPE

Cicluri : 1 sau 3
St ri : 5 sau 11 (6 sau 12 la 8085)
Adresare: registru indirect
Flaguri afectate : nu

RPO   Retur dacă paritatea este impară

Paritatea este impară dacă baitul din acumulator conţine un număr par de biţi de valoarea 1. Această stare este indicată prin condiţia P = 0.

Instrucţiunile RPO şi RPE sînt folosite pentru testarea parităţii datelor de intrare. Printre altele o instrucţiune IN nu trebuie să modifice condiţiile flag.

Pentru a se evita alternarea datelor condiţiile flag pot fi modificate prin adunarea valorii 00H la conţinutul acumulatorului.

Instrucţiunea RPO testează condiţia de paritate P. Dacă P = 0 indicîndu-se astfel o paritate impară, instrucţiunea RPO extrage doi baiţi din stivă şi îi plasează în registrul contorului de program PC. Programul continuă execuţia cu noua adresă din PC. Dacă P = 1 programul execută următoarea instrucţiune secvenţială.

Codul de operare — Operandul

RPO

Nici-un operand nu este admis în instrucţiunea RPO.

EOH = | 1 1 1 0 0 0 0 0 | RPO

Cicluri : 1 sau 3
Stări : 5 sau 11 (6 sau 12 la 8085)
Adresare: registru indirect
Flaguri afectate : nu

RRC — **Rotire dreapta acumulator**

Instrucţiunea RRC face bitul CY egal cu bitul de pon - dere inferioară al acumulatorului. Instrucţiunea deplasează apoi cu un bit spre dreapta toţi biţii din acumulator iar bi - tul de pondere inferioară, după deplasare, este scos şi trecut în poziţia bitului de pondere superioară din acumulator.

Codul de operare — Operandul

RRC

Nici-un operand nu este admis în instrucţiunea RRC.

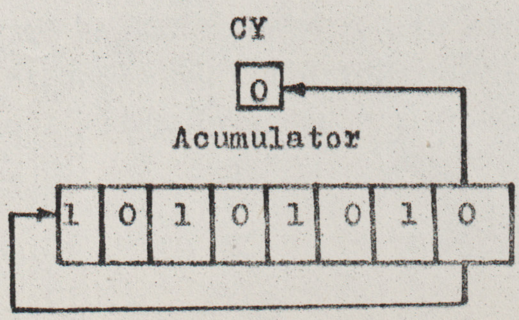
OFH = | 0 0 0 0 1 1 1 1 | RRC

Cicluri : 1
Stări : 4
Flaguri afectate : numai CY

Exemplu :

Se presupune că acumulatorul conţine valoarea OAAH şi CY = 0. In diagrama următoare este ilustrat efectul pe care îl are instrucţiunea RRC asupra acumulatorului.

Inainte:



CY

0

Acumulator

După :							
0	1	0	1	0	1	0	1

RST **Restart**

Instrucţiunea RST are un scop special ca şi instrucţiunea CALL, fiind desemnată în primul rînd pentru a fi folosită la întreruperi.

Instrucţiunea RST introduce în stivă conţinutul registrului contorului de program, acesta formînd adresa de retur şi face apoi un salt la una din cele opt adrese predeterminate. Trei biţi (8 adrese) conţinuţi în baitul codului de operare al instrucţiunii RST, specifică adresa unde trebuie făcut saltul.

Instrucţiunea restart este unică deoarece ea apare rar în aplicaţii. In special perifericele au nevoie de servi - ciul de întreruperi.

Cînd un periferic cere serviciul de întreruperi şi întreruperea este acceptată, procesorul recunoaşte cererea şi pregăteşte magistrala de date pentru a accepta instrucţiunea de un bait de la periferic.

Procesorul mută codul adresei formată din trei biţi din instrucţiunea RST în poziţiile biţilor 3,4 şi 5 din registrul contorului de program PC. Prin aceasta se produce o multiplicare cu 8 a codului. Astfel codul adreselor 000...111 (adică 0...7) în PC va avea adresele 0000H,0008H,0010H,0018H,0020H, 0028H, 0030H şi 0038H .

(Exemplu : $7^{*}8 = 56 = 0038H$)

Programul reia execuţia de la noua adresă dacă este disponibilă pentru serviciul de întreruperi.

Microprocesorul 8085 este prevăzut cu patru intrări hardware care generează intern instrucţiunea RST. In loc să transmită instrucţiunea RST, perifericul aplică doar un semnal de intrare către RST5.5 , RST6.5, RST7.5 sau TRAP şi procesorul generează o instrucţiune RST internă.

Execuţia deplinde de intrare :

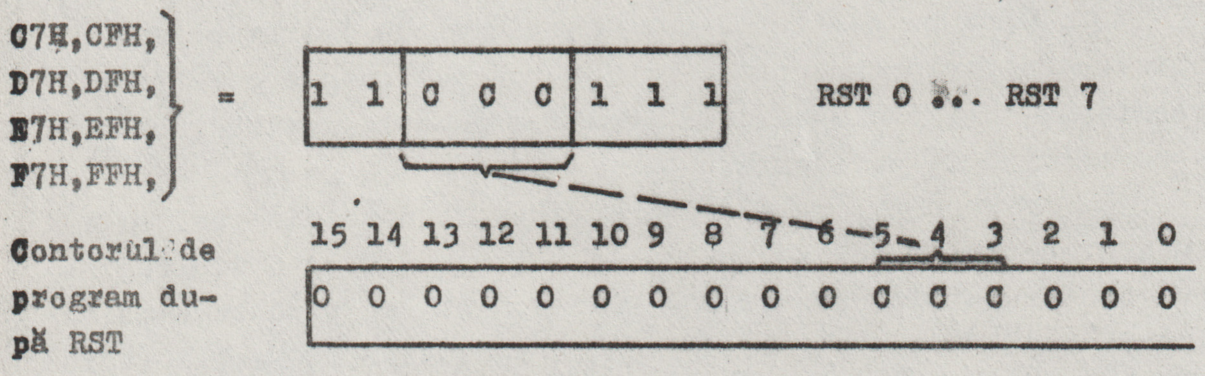
Denumirea intrări	Adresa de restart
TRAP	24H
RST 5.5	2CH
RST 6.5	34H
RST 7.5	3CH

De notat că adresele se găsesc în aceeaşi porţiune de memorie, folosită de instrucţiunea RST astfel că în rutina serviciului de întreruperi sînt suficienţi numai patru biţi pentru CALL, JMP şi RETURN.

Dacă este inclus codul program, instrucţiunea RST are următorul format :

Codul de operare	Operandul
RST	cod adresă

Codul adresei trebuie să fie un număr sau o expresie în domeniul 000B la 111B.



Cicluri : 3

Stări : 11 (12 la 8085)

Adresare: registru indirect

Flaguri afectate : nu

RZ **Retur dacă este zero**

Instrucţiunea RZ testează condiţia Z. Dacă Z = 1 ceea ce înseamnă că toţi biţii din acumulator au valoarea zero, instrucţiunea RZ extrage doi baiţi de date din stivă şi îi plasează în registrul contorului de program PC. Programul continuă execu-

ţia cu o nouă adresă din PC. Dacă Z = 0 programul execută următoarea instrucţiune secvenţială.

Codul de operare	Operandul
RZ	

Nici-un operand nu este permis în instrucţiunea RZ.

C8H = | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | RZ

Cicluri : 1 sau 3
Stări : 5 sau 11 (6 sau 12 la 8085)
Adresare : registru indirect
Flaguri afectate : nu

SBB **Scădere cu împrumut**

Instrucţiunea SBB scade un bait de date şi pe CY = 1 (dacă există) din acumulator şi modifică bitul CY, după scădere.

Rezultatul este stocat în acumulator.

Instrucţiunea SBB foloseşte condiţia CY pentru a permite programului să efectueze scăderea unei serii multibaiţi.

Instrucţiunea SBB încorporează pe CY prin adunarea acestuia la baitul ce trebuie să fie scăzut din acumulator şi execută această scădere prin adunarea în complement de doi.

Scăderea cu împrumut a registrului de acumulator

Codul de operare	Operandul
SBB	reg

Operandul trebuie să specifice unul din registrele A la E, H sau L.

Instrucţiunea SBB scade conţinutul registrului specificat în instrucţiune şi pe CY din acumulator şi stochează rezultatul în acumulator.

98H,99H, 9AH,9BH, 9CH,9DH, 9FH, } = | 1 0 0 1 1 | S S S | { SBB B,SBB C, SBB D,SBB E, SBB H,SBB L SBBA,

Cicluri : 1
Stări : 4
Adresare: registru
Flaguri afectate : Z,S,P,CY,AC

<u>Scăderea cu împrumut a memoriei din acumulator</u>

<u>Codul de operare</u>	<u>Operandul</u>
SBB	M

Instrucţiunea scade pe CY plus conţinutul locaţiei din memorie adresată de registrul pereche H şi L, din acumulator şi stochează rezultatul în acumulator.

9EH = | 1 0 0 1 1 1 1 0 | SBB M

Cicluri : 2
Stări : 7
Adresare: registru indirect
Flaguri afectate : Z,S,P,CY,AC

Exemplu :

Se presupune că registrul B conţine 02H, acumulatorul conţine 04H şi CY = 1. Instrucţiunea SBB B operează în felul următor :

02H + CY = 03H	=	00000011 B
Complementul unu	=	11111100 B
Complementul faţă de doi	=	11111101 B
Acumulatorul 04H	=	00000100 B
		+ 11111101
		1 00000001 = 01H

De notat că la adunarea complementului de doi se produce un transport CY = 1. In acest caz instrucţiunea complementează pe CY astfel că după operare se obţin următoarele condiţii :

CY = 0
S = 0
Z = 0
P = 0
AC = 1

SBI Scădere cu împrumut imediat

Instrucţiunea SBI scade conţinutul celui de al doilea bait al său şi pe CY din conţinutul acumulatorului, Rezultatul este stocat în acumulator.

Instrucţiunea SBI foloseşte condiţia CY pentru a permite programului să efectueze scăderea unei serii multibaiţi.

Instrucţiunea SBI încorporează pe CY prin adunarea acestuia la baitul ce trebuie scăzut din acumulator şi execută scăderea prin adunarea complementului doi.

Asamblorul se caracterizează prin tratarea tuturor simbolurilor externe şi relocabile cu adrese de 16 biţi.

Cînd unul din aceste simboluri apare în expresia operandului instrucţiunei directe, ea trebuie să fie precedată de unul din operatorii HIGH sau LOW care să precizeze care din baiţii adresei trebuie să fie folosit în evaluarea expresiei.

Cînd lipseşte operatorul asamblorul consideră un operator LOW şi dă un mesaj de eroare.

Codul de operare	Operandul
SBI	dată

Operandul trebuie să specifice datele ce vor fi scăzute. Aceste date pot fi un număr, o constantă ASCII, o etichetă a cărei valoare a fost definită anterior, sau o expresie.

Datele nu pot depăşi un bait.

1	1	0	1	1	1	1	0

Cicluri : 2
Stări : 7
Adresare: imediată
Flaguri afectate : Z,S,P,CY,AC

Exemplu :

Această secvenţă de instrucţiuni înlocuieşte mulţimea baiţilor dintr-o arie de 20 de baiţi de la locaţia simbolică AXLOTL cu o mulţime logică formată din unul şi zero în felul următor :

- dacă un element al lui AXLOTL este egal sau mai mare ca 5, în valoare absolută, este înlocuit cu 1 ;
- dacă un element al lui AXLOTL este mai mic ca 5, în valoare absolută, esre inlocuit cu zero.

De notat că desfăşurarea programului este controlată de CY.

```
        MVI B,20       ; iniţializează controlul
        XRA A          ; sterge acumulatorul şi transportul
        LXI H,AXLOTL   ; (H,L) indică mulţimea AXLOTL
LOAD:   MOV A,M        ; încarcă acumulatorul cu baitul indicat
                         de H,L
        SBI 5          ; scade 5,dacă acumulatorul este mai mic
                         ca 5 face CY=1
        JC SMALL       ; salt la SMALL dacă acumulatorul este mai
                         mic ca 5
        MVI M, 1       ; stochează 1 unde a fost mulţimea elemen-
                         telor
        JMP TEST       ; salt înapoi la contorul de test
SMALL:  MVI M,O        ; stochează zero unde a fost mulţimea
                         elementelor
TEST:   XRA A          ; sterge acumulatorul şi transportul
        DCR B          ; decrementează controlul
        CMP B          ; compară B cu zero
        JZ DONE        ; dacă acumulatorul este zero,s-a terminat
                         operaţia
        INX H          ; incrementează H,L pentru a indica următo-
                         rul element din mulţime
        JMP LOAD       ; inapoi şi preia un alt element din mulţime
DONE :                 ; restul programului
```

<u>SHLD</u> <u>Stochează H şi L direct</u>

Instrucţiunea SHLD stochează o copie după conţinutul registrului L la adresa din memorie specificată de baiţii doi şă

trei ai instrucţiunii. După aceasta instrucţiunea stochează o copie după conţinutul registrului H în locaţia imediat următoare a memoriei.

Instrucţiunea SHLD este una din instrucţiunile prevăzute pentru salvarea (păstrare) conţinutului registrelor H şi L. De asemenea, datele din registrele H şi L pot fi plasate în registrele D şi E (instrucţiunea XCHG) sau plasate în stivă (instrucţiunile PUSH şi XTHL).

22 H =	0 0 1 0 0 0 1 0	SHLD adr
	baitul inferior adresă	
	baitul superior adresă	

Cicluri : 5
Stări : 16
Adresare : directă
Flaguri afectate : nu

Exemplu :

Se presupune că registrele H şi L conţin OAEH şi respectiv 29H. In cele ce urmează este ilustrat efectul pe care îl are instrucţiunea SHLD 10AH :

Adrese de memorie :	...109	10A	10B	10C ...
Conţinutul înainte de operarea instrucţiunii SHLD:	... 00	00	00	00 ...
Conţinutul după operare	... 00	29	AE	00 ...

SIM (Numai pentru procesorul 8085) Set masca de întrerupere

Instrucţiunea SIM multiscop foloseşte conţinutul curent al acumulatorului pentru a îndeplini următoarele funcţiuni ;

- setează masca de întrerupere pentru RST 5.5, RST 6,5 şi RST 7,5 la cererea de întrerupere făcută de hardware,; resetează bitul corespunzător lui RST 7.5; dă la ieşire bitul 7 al acumulatorului către interfaţa serială de date.

Codul de operare   Operandul

SIM

Nu este permis nici-un operand în instrucţiunea SIM.

Instrucşiunea SIM interpretează biţii din acumulator în felul următor :

7	6	5	4	3	2	1	0
SOD	SDE	XXX	R7.5	MSE	M7.5	M6.5	M5.5

M7.5, M6.5, M5.5:
Masca pentru RST 7.5
Masca pentru RST 6.5
Masca pentru RST 5.5
0=disponibil
1=mascat

MSE → Permite setul de mascare

R7.5 → Reset RST 7.5: dacă e 1, bitul corespunzător lui RST 7.5 e resetat (devine 0)

XXX → Ignorat

SDE → Dacă e 1, bitul 7 e trimis la ieşirea serială

SOD → Data ieşirii seriale: e ignorată dacă bitul 6=0

SPHL   Transferă H şi L în SP

Instrucţiunea SPHL, încarcă conţinutul registrului pereche H şi L în registrul pointer de stivă SP.

Codul de operare   Operandul

SPHL

Operanzii nu sînt permişi în instrucţiunea SPHL.

Pointerul de stivă SP este un registru cu utilizare specială folosit pentru adresarea stivei; stiva trebuie să fie o memorie cu acces aleator (RAM).

Deoarece diferitele aplicaţii folosesc configuraţii diferite de memorie, utilizatorul programului trebuie să încarce registrul SP cu adresa de început a stivei.

Practic stiva este asigurată în locaţiile disponibile superioare ale memoriei RAM.

Hordware-ul decrementează pointerul de stivă pe măsură ce datele sînt introduse în stivă şi incrementează pointerul cînd

datele sînt scoase.

Pointerul de stivă trebuie să fie inițializat înainte de operarea unei instrucțiuni care are acces la stivă.

F9H =	1 1 1 1	1 0 0 1	SPHL

Cicluri : 1
Stări : 5 (6 la 8085)
Adresare: registru
Flaguri afectate : nu

STA Stochează acumulatorul direct

Instrucțiunea STA stochează o copie după conținutul curent al acumulatorului în locația din memorie care a fost specificată de baiții doi și trei ai instrucțiunii.

Codul de operare	Operandul
STA	adresa

Adresa poate fi formată dintr-un număr, o etichetă anterior definită sau o expresie.

Asamblorul inversează baiții adresei, cel superior cu cel inferior, cînd este formată instrucțiunea.

32H =	0 0 1 1 0 0 1 0	STA addr
	baitul inferior adresă	
	baitul superior adresă	

Cicluri : 4
Stări : 13
Adresare: directă
Flaguri afectate : nu

Exemplu :

Următoarea instrucțiune stochează o copie după conținutul acumulatorului în locația 5B3H din memorie.

STA 5B3H

Cînd este asamblată, instrucțiunea de mai sus are în hexazecimal valoarea 32B305. De notat că asamblorul inversează

baiţii adresei, B3 ou 05, pentru stocarea corectă în memorie.

STAX Stochează acumulatorul indirect

Instrucţiunea STAX stochează o copie după conţinutul acumulatorului în locaţia din memorie care a fost adresată de registrele pereche B sau registrele pereche D.

Codul de operare	Operandul
STAX	B
	D

Operandul B specifică perechea de registre B şi C iar operandul D specifică perechea de registre D şi E.

Această instrucţiune nu poate specifica alte registre în afară de registrele pereche B şi C sau D şi E.

r = 0. 02H = | 0 0 0 | r | 0 0 1 0 | STAX B
r = 1 12H STAX D

0 = registrul pereche B
1 = registrul pereche D

Cicluri : 2
Stări : 7
Adresare : registru indirect
Flaguri adectate : nu

Exemplu :

Dacă registrul B conţine 3FH şi registrul C conţine 16H, următoarea instrucţiune stochează o copie după conţinutul acumulatorului în locaţia 3F16H din memorie : STAX B

STC Modifica transportul : CY = 1

Instrucţiunea STC modifică condiţia CY = 0 şi CY = 1. Celelalte condiţii Z,S,P şi AC rămîn neschimbate.

Codul de operare	Operandul
STC	

Operanzii nu sînt admişi în instrucţiunea STC.

37H = | 0 0 1 1 0 1 1 1 | STC

Cicluri : 1
Stări : 4
Flaguri afectate : CY

<u>SUB</u> <u>Scădere</u>

Instrucţiunea SUB execută scăderea unui bait de date din conţinutul acumulatorului. Rezultatul este stocat în acumulator. Instrucţiunea foloseşte complementul de doi.

De notat că instrucţiunea SUB exclude condiţia CY existentă dar modifică pe CY dacă rezultă un transport după operaţie.

<u>Scăderea registrului din acumulator</u>

<u>Codul de operare</u>	<u>Operandul</u>
SUB	reg

Operandul trebuie să specifice unul din registrele A la E,H sau L. Instrucţiunea scade conţinutul registrului specificat din conţinutul acumulatorului, folosind complementul doi. Rezultatul este stocat în acumulator.

90H, 91H, 92H, 93H, 94H, 95H, 97H = | 1 0 0 1 0 | S S S |

Cicluri : 1
Stări : 4
Adresare : registru
Flaguri afectate : Z,S,P,CY,AC

SUB B, SUB C, SUB D, SUB E, SUB H, SUB L, SUB A

<u>Scăderea memoriei din acumulator</u>

<u>Codul de operare</u>	<u>Operandul</u>
SUB	M

Această instrucţiune scade conţinutul locaţiei din memorie adresată de registrele pereche H şi L. din conţinutul acumulatorului şi stochează rezultatul în acumulator. M este un simbol de referire la registrele H şi L.

96H = | 1 0 0 1 0 1 1 0 | SUB M

Cicluri : 2
Stări : 7
Adresare: registru indirect
Flaguri afectate : Z,S,P,CY,AC

Exemple :

Se presupune că acumulatorul conţine 3EH. Instrucţiunea SUB A scade conţinutul acumulatorului tot din acumulator şi re - zultatul trebuie să fie zero care rămîne stocat în acumulator în felul următor :

```
                3EH = 00111110
            +(-3EH) = 11000001 complement de unu la care
                             1 se adună 1 şi rezultă complemen-
                      -------- tul de doi
Se  anulează CY=1    00000000
```

Condiţiile flag rezultate după operare sînt următoarele:

C = 0
S = 0
Z = 1
P = 1
AC = 1

De notat că transportul auxiliar AC este 1 deoarece în exemplul dat a existat un transport după bitul 3.

<u>SUI</u> <u>Scădere imediată</u>.

Instrucţiunea SUI scade conţinutul celui de al doilea bait al său din acumulator şi stochează rezultatul în acumulator. De notat că instrucţiunea SUI neglijează transportul CY existent (la scădere CY = împrumut) pe durata scăderii,dar reconsideră pe CY după operaţie.

<u>Codul de operare</u>	<u>Operandul</u>
SUI	data

Operandul trebuie să conţină datele ce vor fi scăzute. Aceste date pot fi un număr, o constantă ASCII, o etichetă definită anterior, sau o expresie. Datele nu pot depăşi un bait.

Asamblorul se caracterizează prin tratarea tuturor sim-

bolurilor externe şi relocatabile ca adrese de 16 biţi.

Cînd unul din aceste simboluri apare în expresia operandului instrucţiunii imediate, el trebuie să fie precedat de unul din operatorii HIGH sau LOW care să precizeze care din baiţii adresei trebuie să fie folosit în evaluarea expresiei.

Cînd lipseşte operatorul asamblorul consideră un ope - rand LOW şi dă un mesaj de eroare.

D6H = | 1 1 0 1 0 1 1 0 | SUI date

Cicluri : 2

Stări : 7

Adresare : imediată

Flaguri afectate : Z,S,P,CY,AC

Exemplu :

Se presupune că acumulatorul conţine valoarea 9 cînd este executată instrucţiunea SUI 1.

Acumulatorul	= 00001001	= 9H
Date directe (compl 2)	= 11111111	=-1H
	100001000	= 8H

Se modifică CY = 1 în CY = 0

Indicatorii de condiţie rezultaţi după operaţie sînt :

CY = 0
S = 0
Z = 0
P = 0
AC = 1

<u>XCHG</u> <u>Schimbă H şi L cu D şi E</u>

Instrucţiunea XCHD schimbă conţinutul registrului pereche H şi L cu conţinutul registrului pereche D şi E.

<u>Codul de operare</u>	<u>Operandul</u>
XCHD	

Operanzii nu sînt admişi în instrucţiunea XCHG.

Instrucţiunea XCHG salvează conţinutul existent în registrul pereche H şi încarcă o nouă adresă în acest registru.

Instrucţiunea XCHG este o instrucţiune registru la registru, oferind mijlocul cel mai rapid de salvare şi/sau alterare a registrului dublu H,L.

EBH = | 1 1 1 0 1 0 1 1 | XCHD

Cicluri : 1

Stări : 4

Adresare : registru

Flaguri afectate : nu

Exemplu :

Se presupune că registrul pereche H şi L conţine 1234H şi registrul pereche D şi E conţine OABCDH. După executarea instrucţiunii XCHG,H şi L va conţine OABCDA iar registrul pereche D şi E va conţine 1234H.

XRA SAU exclusiv cu acumulatorul

Instrucţiunea XRA execută operaţiunea logică SAU exclusiv între conţinutul baitului spe i icat şi conţinutul acumulatorului. Rezultatul este plasat în acumulator.

Instrucţiunea XRA între registru şi acumulator

Codul de operare	Operandul
XRA	reg

Operandul trebuie să specifice unul din registrele A la E H şi L. Această instrucţiune execută o operaţiune logică SAU exclusiv folosind conţinutul registrului specificat şi conţinutul acumulatorului, după care stochează rezultat 1 în acumulator.

Indicatorii de condiţie CY şi AC sînt trecute în zero.

A8H, A9H, AAH, ABH, ACH, ADH, AFH = | 1 0 1 0 1 | S S S |

Cicluri : 1

Stări : 4

Adresare: registru

Flaguri afectate:Z,S,P,CY,AC

XRA B
XRA C
XRA D
XRA E
XRA H
XRA L
XRA A

Instrucţiunea XRA între memorieşi acumulator

Codul de operare	Operandul
XRA	M

Această instrucţiune execută o operaţiune logică SAU exclusiv folosind conţinutul locaţiei din memorie specificată de registrul pereche H şi L şi conţinutul acumulatorului, după care stochează rezultatul în acumulator.

AEH = | 1 0 1 0 1 1 1 0 | XRA M

Cicluri : 2

Stări : 7

Adresare: registru indirect

Flaguri afectate: Z,S,P,CY,AC

Exemplu :

Deoarece operaţiunea logică SAU exclusiv dintre un bit cu el însăşi produce întodeauna un zero, instrucţiunea XRA este frecvent folosită pentru a aduce acumulatorul în zero. Următoarele instrucţiuni aduc în zero acumulatorul şi registrele B şi C.

```
XRA A
MOV B,A
MOV C,A
```

Operaţiunea logică SAU exclusiv, dintre un bit oarecare (stare 1 sau 0) cu un bit stare 1 întodeauna complementează bitul oarecare. Astfel, dacă acumulatorul conţine toţi biţii de valoare 1 adică OFFH, instrucţiunea XRA B produce complementul de unu al registrului B în acumulator.

XRI SAU exclusiv imediat cu acumulatorul

Instrucţiunea XRI execută o operaţiune logică SAU exclusiv între conţinutul celui de al doilea bait al său şi conţinutul acumulatorului.

Rezultatul este plasat în acumulator.

Instrucţiunea XRI trece în zero condiţiile CY şi AC.

Codul de operare	Operandul
XRI	date

Operandul trebuie să specifice datele care vor di folo - site în operaţiunea logică SAU exclusiv. Aceste date pot avea forma unui număr, o constantă ASCII, o etichetă a cărei valoare a fost definită anterior, sau o expresie.

Datele nu pot depăşi un bait (8 biţi).

Asamblorul se caracterizează prin tratarea tuturor sim - bolurilor externe şi relocatabile cu adrese de 16 biţi.

Cînd unul din aceste simboluri apare în expresia operandului instrucţiunii, directe, ea trebuie să fie precedată de unul din operatorii HIGH sau LOW care să precizeze care din baiţii de adresă trebuie să fie folosit în evaluarea expresiei.

Cînd lipseşte operatorul, asamblorul consideră un operator LOW şi dă un mesaj de eroare.

EEH =	1	1	1	0	1	1	1	0	XRI
	date								

Cicluri : 2
Stări : 7
Adrèsare: imediată
Flaguri afectate : Z,S,P,CY,AC

Exemplu :

Se presupune că programul foloseşte biţii 7 şi 6 dintr-un bait ca biţi de control pentru apelarea a două subrutine.

Programul testează starea acestor biţi prin rotirea conţinutului acumulatorului astfel ca bitul dorit să ajungă în CY, după care instrucţiunea CC (call if carry) testează indicatorul de condiţie CY şi apelează subrutina dacă CY = 1.

Se presupune că baitul ce conţine biţii de control este normal poziţionat în acumulator şi că programul trebuie să pună bitul 6 în 0 şi bitul 7 în 1.

Restul biţilot 0 ... 5 care sînt folosiţi pentru a controla alte sarcini nu trebuie să fie modificaţi.

Intrucît operaţiunea logică SAU exclusiv dintre un bait cu el însuşi produce întodeauna un zero, iar operaţiunea dintre un bit oarecare cu un bit 1 complementează întodeauna bitul oarecare, se foloseşte următoarea instrucţiune :

XRI 11000000B

Această instrucţiune are asupra baitului din acumulator următorul efect :

		76543210
Acumulatorul	=	01001100
Datele directe	=	11000000
		10001100

Se poate vedea că instrucţiunea a trecut bitul 7 în ON (unu) şi bitul 6 în OFF (zero) iar restul biţilor 0...5 au rămas neschimbaţi.

XTHL Schimbă H şi L cu vîrful stivei

Instrucţiunea XTHL schimbă doi biţi aflaţi în vîrful stivei cu cei doi baiţo conţinuţi în registrul pereche H şi L. Pentru aceasta instrucţiunea salvează conţinutul existent al registrului pereche H şi L şi încarcă noua valoare,în această pereche de registre.

Codul de operare	Operandul
XTHL	

In instrucţiunea XTHL nu este admis nici-un operand. Instrucţiunea XTHL schimbă conţinutul registrului L cu conţinutul locaţiei din memorie specificată de registrul SP (Stack Pointer) şi apoi conţinutul registrului H cu conţinutul locaţiei din memorie specificată de SP + 1.

E3H = | 1 1 1 0 0 0 1 1 | XTHL

Cicluri : 5
Stări : 18 (16 la 8085)
Adresare: registru indirect
Flaguri afectate : nu

Exempl

Se presupune că registrul SP conţine adresa 10ADG registrul H con'ine OBH şi registrul L conţine 3CH.

Locaţiile din memorie 10ADH şi 10AEH conţin FOH şi respectiv ODH.

In cele ce urmează este ilmstrat efectul pe care îl are executarea instrucţiunii XTHL :

	ADRESE DE MEMORIE				H	L
	10AC	10AD	10AE	10AF		
Conţinutul înainte de XTHL	FF	FO	OD	FF	OB	3C
Conţinutul după XTHL	FF	3C	OB	FF	OD	FO

Registrul SP rămîne neschimbat după executarea instrucţiunii XTHL.

4. PSEUDOINSTRUCTIUNI, DECLARATII

(Assembler Directives)

Acest capitol descrie pseudoinstrucţiunile folosite în comanda asamblorului 80/85 în generarea programului obiect (în cod maşină). Acest capitol exclude macroinstrucţiunile care vor fi tratate separat în cap.5.

In general pseudoinstrucţiunile au aceeaşi formă ca instrucţiunile şi pot fi interschimbate în program.

Gruparea pseudoinstrucţiunilor :

Pseudoinstrucţiuni generale

- Definire simbol
    - EQU
    - SET
- Definirea datelor
    - DB
    - DW
- Rezervare memorie
    - DS
- Asamblare condiţionată
    - IF
    - ELSE
    - ENDIF
- Sfîrşit de asamblare
    - END

Declaraţii

- Definirea adresei locaţiei
    - ASEG
    - DSEG
    - CSEG
    - ORG
- Linkarea programului
    - PUBLIC
    - EXTRN
    - NAME
    - STKLN

Trei pseudonstrucţiuni - EQU, SET şi MACRO - au format diferit de instrucţiunile limbajului de asamblare. Pseudoins - trucţiunile mai sus amintite necesită un nume pentru simbolul sau macroul, definite ca prezente în cîmpul etichetei. Numele diferă de etichete prin faptul că nu se pot termina cu două puncte ca etichetele. De asemenea pseudoinstrucţiunile ENDM şi LOCAL interzic utilizarea cîmpului - etichetă.

Definirea simbolurilor

Asamblorul asignează valori simbolurilor care apar în cîmpul etichetă.

Această valoare este starea curentă a numărătorului de adresă (location counter) cînd instrucţiunea este asamblată.

Se pot defini alte simboluri şi li se pot asig a valori folosind EQU şi SET.

Simbolurile definite cu EQU nu mai pot fi redefinite în timpul programului ; celor definite prin SET li se pot asigna noi valori cu SET-uri ulterioare.

Numele din cîmpul etichetă al lui EQU şi SET nu se pot termina cu două puncte (":")!

Simbolurile definite prin EQU şi SET rămîn valabile pe tot restul programului. Acest fapt poate cauza definiri multi - ple ilegale cînd EQU şi SET apar în macrodefiniţii. Cu ajutorul instrucţiuni LOCAL se poate evita această problemă (vezi cap.5).

Pseudoinstrucţiunea EQU

EQU asignează valoarea "expresiei" numelui din cîmpul etichetă.

Etichetă (Label)	Mnemonic (Opcode)	Operand
nume	EQU	expresie

Expresia EQU nu poate conţine nici un simbol exterior. (Simbolulexterior va fi tratat mai tîrziu).

In momentul evaluării expresiilor EQU, întotdeauna se generează o adresă modulo 64K. Deci, expresia întotdeauna livrează valori în domeniul 0 - 65535.

Exemplu :

Următorul EQU introduce numele ONES în tabela de simboluri şi îi asignează valoarea binară 11111111

ONES EQU OFFH

Valoarea asignată de EQU poate fi rechemată în linii subsecvente ale programului sursă prin referire la numele asignat lui, ca în următoarea instrucţiune IF (unde IF a fost anterior definit) :

```
IF TYPE EQ ONES
.
.
.
ENDIF
```

Pseudoinstrucţiunea SET

SET asignează valoarea 'expresiei' numelui specificat în cîmpul etichetă.

Etichetă	Mnemonic	Operand
nume	SET	expresie

Asamblorul introduce valoarea 'expresiei' în tabelul simbolurilor. Ori de cîte ori 'numele' este întîlnit ulterior în cursul asamblării, asamblorul substituie acestuia valoarea din tabelul simbolurilor. Această valoare rămîne neschimbată pînă nu este alterată de o pseudoinstrucţiune SET următoare .

Funcţia lui SET e identică cu EQU exceptînd cazul cînd numele poate să apară în multiple pseudoinstrucţiunea SET din acelaşi program. Deci, putem altera valoarea asignată numelui în decursul asamblării. In momentul evaluării expresiei lui SET, întotdeauna se generează o adresă modulo 64 K. Deci expresia întotdeauna livrează valori în domeniul 0-65535.

Etichetă	Mnemonic	Operand	Codul asamblat
IMMED	SET	5	
	ADI	IMMED	C605
UMMED	SET	10H-6	
	ADI	IMMED	C60A

Definirea datelor

DB-ul (definire byte) şi DW-ul (definire cuvînt, în engleză:define word) fac posibilă definirea datelor ce urmează a fi stqcate în program. Datele por fi specificate în cuvinte de 8 sau 16 biţi sau şir de caractere de text.

Pseudoinstrucţiunea DB

DB-ul înmagazinează datele specificate în locaţii consecutive de memorie începînd cu starea curentă a numărătorului de adrese.

Etichetă	Mnemonic	Operanzi
opţional:	DB	expresii sau şiruri sii

Orice simbol din expresii trebuie definit anterior. Cîmpul - operand al DB poate conţine o listă de expresii şi/ sau linii de text. Lista poate conţine pînă la opt unităţi de informaţie întregi; unităţile de informaţie trebuie să fie despărţite prin vitgulă. Din cauza spaţiului de lucru limitat, asamblorul poate să nu fie capabil să manipuleze un total de opt unităţi de informaţie, cînd lista include un număr de expresii complexe. Oricînd se iveşte această problemă se rezolvă uşor : se folosesc două sau mai multe pseudoinstrucţiuni pentru a scurta lista.

Expresiile se evaluează pînă la numere de 8 biţi (1byte) în domeniul - 256 pînă la 255. Sirurile de texte pot conţine maximum 128 caractere ASCII incluse în referiri.

Proprietatea de relocatare a asamblorului tratează toate simbolurile externe şi relocabile ca adrese de 16 biţi.

Cînd unul din aceste simboluri apare în una din expresiile LOW sau HIGH pentru a specifica care din byte-urile adresei trebuie folosite în evaluarea expresiei. Cînd nu este nici un operator prezent, asamblorul îşi atribuie operatorul LOW şi emite un mesaj de eroare.

Dacă eticheta operaţională este prezentă, numărătorului de adrese ie se asignează valoarea de start, aceasta adresînt primul byte stocat de DB. Deci, eticheta STR în exemplele următoare se referă la litera T din şirul TIME :

Etichetă	Mnemonic	Operanzi	Codul asamblat
STR:	DB	'TIME'	54494D45
HERE:	DB	0A3H	A3
WOED1:	DB	-3H,5*2	FD0A

Pseudoinstrucţiunea DW

DW stochează fiecare valoare de 16 biţi din lista de expresii ca o adresă. Valorile sînt stocate începînd cu starea curentă a numărătorului de adresă.

Etichetă	Mnemonic	Operanzi
opţional:	DW	listă de expresii

Fiecare simbol din lista de expresii trebuie definit anterior. Cei mai puţin semnificativi opt biţi ai primei valori din lista de expresii, sînt stocaţi de starea curentă a adresei; cei mai semnificativi opt biţi sînt stocaţi de adresa superioară ce urmează. Acest proces se repetă pentru fiecare unitate de informaţie din lista de exoresii.

Expresia evaluează un cuvînt de 16 biţi, în mod tipic,o adresă. Dacă expresia evaluează un singur byte, se consideră primii opt biţi nesemnificativi ai cuvîntului de 16 biţi, iar biţii cei mai semnificativi sînt toţi zero.

Unităţile informaţionale ale listei se separă prin virgulă. Lista poate conţine pînă la 8 unităţi informaţionale complete. Din cauza spaţiului de lucru limitat asamblorul poate să nu fie capabil să manipuleze un total de 8 expresii complete. Pentru a rezolva acest impact decizional se utilizează două sau mai multe "DW" pentru a scurta lista. Ordinea inversată pentru stocarea byte-ţilor de ordine inferioară şi superioară constituie formatul tipic pentru adresele stocate în memorie. Deci,DW e de obicei folosit pentru stocarea constantelor de adresă.

Şirurile conţinînd 1 sau două caractere ASCII încadrate în ghilimele pot de asemenea să apară în lista de expresii.Cînd se utilizează asemenea şiruri în programul dvs., caracterele sînt stocate în ordine inversă. Trimiţînd un şir mai lung de două caractere cauzează erori. Dacă e prezentă o etichetă opţională numărătorului de adrese i se asignează valoarea de start, acesta adresînd primul byte stocat de DW. (Acesta este byte-ul de ordine inferioară al primei unităţi de informaţie din lista de expresii).

Exemple :

Considerăm că COMP şi FILL sînt etichete definite în altă parte a programului COMP adresează locaţia de memorie

3B1CH; FILL adresează locaţia 3EB4H.

Etichetă	Mnemonic	Operanzi	Cod asamblat
ADDR1:	DW	COMP	1C3B
ADDR2:	DW	FILL	B43E
STRNG:	DW	'A','AB'	41004241
FOUR:	DW	4H	0400

Rezervare de memorie

Pseudoinstrucţiunea DS

DS poate fi utilizat la definirea unui segment de memorie.

Etichetă	Mnemonic	Operand
opţional	DS	expresie

Fiecare simbol al expresiei trebuie să fie definit anterior. Valoarea 'expresiei' specifică numărul byte-ţilor ce vor fi alocaţi pentru memorare de date. Teoretic valorile pot fi 00H la 0FFFFH . In practică nu se alocă mai mult spaţiu decît cel aflat la dispoziţie şi se va lăsa loc pentru program.

Oricare simbol care apare în expresia operanzilor trebuie definit înainte ca asamblorul să ajungă la pseudoinstrucţiunea DS. DS nu asamblează date în program, precum o face DB şi DW. Conţinutul segmentului de memorie alocat este imprevizibil în momentul iniţierii expresiei programului.

Dacă eticheta opţională e prezentă, se asignează valoarea curentă a numărătorului de adresă şi în acest fel se adresează primul byte al blocului de memorie alocat.

Dacă valoarea expresiei operand este zero, nu e nevoie de memorie. Oricum, dacă eticheta opţională e prezentă se asignează valoarea curentă a numărătorului de adresă. DS rezervă memoria prin incrementarea numărului de adresă cu valoarea expresiei operand.

Exemple :

```
TTYBUF: DS  72    ; rezerva 72 byte-ţi pentru
                  ; un BUFFER tampon terminal de ieşire
```

La codarea datelor, se ţine seamă de implementarea ROM, RAM existentă. In general DB şi DW definesc constante memorabile în ROM. Acestea nu se pot modifica. Dacă acestea sînt asignate RAM-ului, ele au o valoare iniţială pe care programul o poate modifica în cursul execuţiei. Valorile iniţiale trebuiesc reîncărcate înaintea fiecărei execuţii de program.

Datele variabile se asignează RAM-ului.

Descrierea datelor

Inaintea codării programului trebuie să se cunoască detaliat datele de intrare şi ieşire. Dar, probabil e mai convenabil a amîna codarea descrierii datelor pînă la dezvoltarea destul de completă a programului care a mai rămas. In acest fel, veţi avea o idee mai clară asupra constantelor şi a domeniilor de lucru necesare programului. De asemenea, organizarea unui program tipit plasează instrucţiunile în memoria inferioară, care este urmată de date şi stivă.

Accesul la date

Accesul la datele din memorie e un proces tipic de doi paşi: I) se spune procesorului unde va găsi data; II) procesorul extrage datele din memorie şi le încarcă în registru, de obicei în acumulator. Deci , următoarele secvenţe de cod au acelaşi efect de încărcare a caracterului ASCII "A" în acumulator.

```
AAA:   DB     'A'      ALPHA:   DB     'ABC'
       .                        .
       .                        .
       LXI    B,AAA             LXI    B,ALPHA
       LDAX   B                 LDAX   B
       .                        .
       .                        .
```

In aceste exemple instrucţiunea LXI încarcă adresa datei dorite în registrele B şi C. Instrucţiunea LDAX încarcă apoi în acumulator un byte al datei de la adresa specificată de registrele B şi C. Asamblorul nici nu ştie nici nu ţine cont de faptul că s-a avut acces la numai unul din caracterele cîmpului de trei caractere ALPHA. Programul trebuie să ţină cont de caracterele ALPHA+1 şi ALPHA+2 ca următoarea secvenţă de cod :

```
ALPHA :   DB      'ABC:      ; DEFINE ALPHA (definire ALPHA)

          LXI     B,ALPHA    ; LOAD ADDRESS OF ALPHA (încarcă
                               adresa lui ALPHA)
          LDAX    B          ; FETCH 1ST ALPHA CHAR (extrage
                               primul caracter)
          INX     B          ; SET B TO ALPHA+1 (fixează pe B
                               la valoarea ALPHA+1)
          LDAX    B          ; FETCH 2ND ALPHA CHAR

          INX     B          ; SET B TO ALPHA+2
          LDAX    B          ; FETCH 3RD ALPHA CHAR
```

Codarea de mai sus e acceptabilă pentru cîmpuri de date scurte ca ALPHA . Pentru cîmpuri mai lungi se poate aloca memorie organizînd o secvenţă de instrucţiuni care se execută în mod repetat pînă la epuizarea sursei de date.

Simboluri suplimentare pentru accesul la date

Exemplul următor e prezentat în prealabil ca o ilustrare a pseudoinstrucţiunii DS.

Etichetă	Mnemonic	Operand	Comentariu
TTYBUF:	DS	72	;rezervă guffer TTY

Accesul la date în acest buffer (tampon) folosind numai expresii ca TTYBUF+1, TTYBUF+2, ... TTYBUF+72 poate fi foarte laborios şi confuz mai ales cînd vreţi numai să selectaţi cîmpuri din tampon. Se poate simplifica această operaţie prin subdiviziunea tamponului cu pseudoinstrucţiuni EQU.

Etichetă	Mnemonic	Operand	Comentariu
TTYBUF:	DS	72	; rezervă tamponul TTX
ID	EQU	TTYBUF	; RECORD IDENTIFIER (identificare înregistrare)
NAME	EQU	TTYBUF+6	; cîmp de nume pentru 20 caractere
NUMBER	EQU	TTYBUF+26	; 10-CHAR EMPLOYEE NUMVER (numele a 10 angajaţi)

Eticheta	Mnemonic	Operand	Comentariu
DEPT	EQU	TTYBUF+36	; 3 caractere pentru numărul departamentului
SSNO	EQU	TTYBUF+41	;numărul sectorului social
DOH	EQU	TTYBUF+50	; data închirierii
DESC	EQU	TTYBUF+56	; descrierea jobului

Subdivizînd datele, ca în exemplul anterior se simplifică accesul la date, furnizează o documentaţie folositoare pe tot cursul programului. De remarcat că acest EQU-uri pot fi inserate oriunde în program după necesităţi, dar codarea lor ca în exem - pul anterior asigură o descriere mai utilitară a informaţiei.

Asamblare condiţionată

Pseudoinstrucţiunea IF, ELSE şi ENDIF fac posibile asamblarea condiţionată a unor părţi din program. acestea însemnînd că asamblarea se realizează numai cînd sînt satisfăcute anumite condiţiuni specifice anterior.

Fiecare simbol în cadrul blocului IF-ENDIF, trebuie de - finit iniţial.

Asamblarea condiţionată e folosită în special atunci cînd aplicaţia necesită programe de largă utilitate pentru un număr de operaţii obişnuite.

De exemplu, presupunînd că un program de control de bază necesită generalizarea pentru a accepta intrarea unuia din 6 echipamente de citire şi pentru a comanda unul din 5 echipamente de control diferite. In loc să se codeze cca 30 de program separate pentru a ţine cont de toate posibilităţile, se poate coda un singur program. Codul pentru senzorii şi driver-ii individu - ali, trebuie să fie conţinut de pseudoinstrucţiunea condiţionată. Cînd este necesară generarea unui program de largă utilitate, se pot însera pseudoinstrucţiunea SET la începutul programului sursă pentru selectarea rutinei senzorului şi driverului dorit.

Pseudoinstrucţiunea IF,ELSE,ENDIF

Deoarece aceste pseudoinstrucţiuni sînt utilizate în interdependenţă, vor fi descrise împreună.

Etichetă	Mnemonic	Operand
opţional :	IF	expresie
opţional :	ELSE	
opţional :	ENDIF	

Fiecare simbol din expresie trebuie definit anterior. Asamblorul evaluează expresia din cîmpul operandului pseudoinstrucţiunii IF. Dacă bitul zero al valorii rezultate este unu (TRUE) toate instrucţiunile între IF şi următorul ELSE sau ENDIF sînt asamblate. Cînd bitul zero este zero /FALSE/ aceste instrucţiuni sînt neglijate. (O expresie adevărată /TRUE/ are valoarea OFFFFH şi una falsă /FALSE/ e OH; numai bitul zero trebuie testat).

Toate afirmaţiile incluse între IF şi ENDIF-ul necesar asociat, sînt definite ca un bloc IF-ENDIF . ELSE e opţională şi numai un singur ELSE poate să apară în blocul IF-ENDIF. Cînd este inclus, ELSE e inversul lui IF. Cînd bitul zero al expresiei lui IF este zero toate afirmaţiile dintre ELSE şi ENDIF următor sînt asamblate. Dacă bitul zero este unu, aceste afirmaţii sînt neglijate.

Operanzii nu sînt permişi împreună cu ELSE şi ENDIF.

Un bloc IF-ENDIF poate să apară în cadrul altui bloc IF-ENDIF. Aceste blocuri sînt grupate pe opt nivele.

Definiţii macro pot să apară în cadrul blocului IF-ENDIF, şi invers. (Blocuri IF-ENDIF pot apărea în marco definiţii). In ambele cazuri trebuie să fim siguri pe terminarea definiţiei macro sau a blocului IF-ENDIF, aşa fel încît să poată fi asamblat complet.

De exemplu dacă definirea unui macro începe într-un bloc IF dar se termină după un ELSE, numai o porţiune din macro poate fi asamblată. In mod similar un bloc IF-ENDIF început în cadrul definirii unui macro trebuie terminat în aceeaşi definire macro.

Notă : Se cere atenţie cînd simbolurile sînt definite în blocuri IF-ENDIF şi se fac referiri la ele în alt loc în program. Aceste simboluri sînt nedefinite cînd evaluarea expresiei IF suprimă asamblarea blocului IF-ENDIF.

Exemplu :

1 bloc IF.ENDIF simplu (unde TYRE a fost anterior definit) :

```
CONDI:      IF TYPE EQ 0
            .
            .                   ; asamblat dacă 'TYPE' =0'
            .                   ; e adevărat
            .
            ENDIF
```

EX.2 IF-ELSE-ENDIF bloc :

```
COND2:      IF TYPE EQ 0
            .                   ; asamblat dacă 'TYPE =0'
            .                   ; E adevărat
            .
            ELSE
            .
            .                   ; asamblat dacă 'TYPE=0'
            ENDIF
```

Exemple 3 IF-uri grupate

```
COD3:       IF TYPE EQ 0
            .
            .                   ; asamblat dacă 'TYPE =0'
            .                   ; e adevărat
            .
            IF MODE EQ 1
            .
LEVEL       .                   ; asamblat dacă 'TYPE=0'
  1         .                   ; si 'MODE=1'amîndăuă sînt
            .                   ; adevărate
            ENDIF
            ELSE
            .
LEVEL       .                   ; adamblat dacă 'TYPE=0'
  2         .                   ; e fals
            .
            IF MODE EQ 2
            .
            .                   ; asamblat dacă 'TYPE=0'
            .                   ; e fals şi 'MODE=2'
            .                   ; e adevărat
```

```
LEVEL   ELSE
  1     .
        .                ; asamblat dacă 'TYPE=1'
        .                ; şi'MODE=2' sînt amîndouă
        .                ; false
        ENDIF
        ENDIF
```

<u>Sfîrşitul asamblării</u>

<u>Pseudoinstrucţiunea END</u>

END idenfitică sfîrşitul programului sursă şi termină fiecare pas de asamblare.

Etichetă	Mnemonica	Operand
opţional :	END	expresie

Numai o singură expresie END poate apărea în programul sursă şi aceasta trebuie să fie ultima expresie a programului sursă.

Dacă expresia opţională e prezentă, valoarea ei este folosită ca adresă de start pentru executarea programului.Dacă nu se dă nici o expresie, asamblorul alocă zero drept adresă de start.

Cînd se asociază mai multe module de program separate, numai într-unul din ele se poate specifica adresa de start. Modulul de program avînd adresa de start este modulul principal. Cînd rîndurile programului sursă sînt combinate utilizînd controlul lui INCLUDE, nu sînt restricţii în ceea ce priveşte rîndul sursei care conţine END.

<u>Indicaţia END-OF-TAPE</u>

EOT facilitează specificarea sfîrşitului fizic al benzii perforate, pentru simplificarea asamblării în cazul programului sursă încărcat pe mai multe benzi.

<u>Pseudoinstrucţiunea EOT</u>

Etichetă	Mnemonic	Operand
opţional:	EOT	...

Cînd EOT e recunoscută de asamblor mesajul 'NEXT TAPE' e trimis la consolă şi asamblorul aşteaptă (e în pauză). După ce banda următoare e încărcată, un blanc primit la consolă semnalizează continuarea asamblării.

Date în cîmpul operand cauzează eroare.

Controlul numărătorului de adrese şi realocarea

Toate pseudoinstrucţiunile discutate în partea următoare a acestui cap. se referă direct la relocarea programului exceptînd pseudoinstrucţiunea ASEG şi ORD. Aceste pseudoinstrucţiuni sînt descrise mai întîi pentru utilizatorii care nu folosesc posibilitatea de locatare.

Controlul numărului de adresă (Model nerelocatabil)

Cînd se alege modul nerelocatabil, un defect al asamblorului va genera pseudoinstrucţiunea ASEG. ASEG-ul arată că programul e asamblat în mod nerelocatabil şi alocă un numărător de adrese pentru asamblare.

Numărătorul de adrse realizează aceeaşi funcţie pentru asamblor ca şi următorul de program în timpul executării programului. El arată asamblorului următoarea locaţie de memorie disponibilă pentru asamblare de date sau instrucţii.

Iniţial numărătorul de adrese e adus în poziţie zero. Adresa poate fi modificată de ORG (origine).

Pseudoinstrucţiunea ORG

ORG pune numărătorul de adrse la valoarea specificată de expresia operandului.

Etichetă	Mnemonic	Operand
opţional:	ORG	expresie

Evaluarea expresiei ORG în timpul asamblării, generează întodeauna o adresă modulo 64K. Deci, expresia întodeauna generează o adresă în domeniul 0 la 65.535. Fiecare simbol al expresiei trebuie definit anterior. Următoarea instrucţiune în cod maşină sau unitate de informaţie e asamblată la adresa specificată.

Dacă nu e inclus ORG înaintea primei instrucţiuni sau byte de date din program, asamblarea poate include oricîte pseudoinstrucţiuni ORG. Mai multe ORG nu necesită specificarea

adreselor în ordine crescătoare, dacă totuşi se omite acest lucru, trebuie ordonat asamblorului să supraînscrie pe o porţiune de program dinainte asamblat.

Dacă eticheta opţională e prezentă, se asignează valoarea curentă numărătorului de adrese înainte ca ORG să fi actualizat numărătorul.

Exemplu :

Presupunem că valoarea curentă a numărătorului de adrese este OFH (zecimal 15) cînd următorul ORG e întîlnit

```
PAG1 :   ORG    OFFH    ; asamblorul ORG la locaţia
                        ; OFFH (zecimal 225)
```

Simbolul PAG1 e asignat adresei OFH. Următoarea instrucţiune sau byte de date e asamblat la locaţia OFFH.

Introducere în modul relocatabil

O caracteristică importantă a unui asamblor este sistemul său generator de module de cod maşină relocatabile. Suportul acestei proprietăţi îl constituie un număr de noi pseudoinstrucţiuni pentru asamblor şi trei noi programe incluse în SISIS-II. Cele trei noi programe - LIB,LINK şi LOCATE sînt descrise în Ghidul de utilizare a sistemului ISIS-II. Noile pseudoinstrucţiuni ale asamblorului urmează a fi descrise în acest capitol.

Relocatarea facilitează programatorului codarea programelor sau secţiunilor de program, fără a ţine cont de amplasamentul final al programului obiect în memorie. Aceasta oferă celor ce dezvoltă sisteme de microcalculatoare avantaje majore în două domenii : gestionarea memoriei. dezvoltarea modulară a programului.

Gestionarea memoriei

Cînd dezvoltăm, testăm sau depănăm un sistem în cadrul sistemului de dezvoltare cu microcalculatorul,Intellec, toată greutatea locatării programului este că acesta să nu se suprapună rutinelor rezidente ISIS-II.

Deoarece sistemul Intellec are 32K,48K,64K octaţi RAM, adresarea programului nu prezintă dificultăţi. Totuşi programul pe care îl dezvoltaţi desigur va conţine mixturi de RAM,ROM,şi/ sau PROM.

Proprietatea de relocatare facilitează dezvoltarea, testarea şi depanarea programului pe sistemul de dezvoltare Intellec,şi după aceea se procedează la relocatarea codului maşină în aşa fel, încît să corespundă aplicaţiei.

Relocatarea mai are un avantaj major pe parcursul asamblării : adesea, programe mari cu multe simboluri nu pot fi a - samblate din cauza spaţiului de lucru limitat al tabelului de simboluri. Un asemenea program poate fi divizat în mai multe module asamblabile separat, şi apoi legate împreună într-un singur program obiect.

Dezvoltarea modulară a programului

Importanţa primordială a relocatării este posibilitatea subdivizării unui program complex într-o serie de programe mai mici şi mai simple.

Exemplu :

Presupunem că un program de microcalculator controlează aprinderea unui motor de automobil. Aceasta cere programului: controlul temperaturii ambiante, temperatura aerului absorbit de motor, temperatura de răcire, valoarea depresiunii din motor, detecţia mersului în dol, determinarea sarcinii motorului. Se examinăm două posibilităţi de rezolvare a acestei probleme.

In ambele cazuri trebuie calculată,proporţia avansului aprinderii, sau a întîrzierii, care face posibil un consum minim de combustibil şi poluare minimă. Primul program (A) citeşte toate intrările şi calculează avansul corect al aprinderii. Programul B foloseşte o aproximare modulară şi codează programe separate pentru fiecare intrare plus un program pentru calcularea avansului aprinderii.

Programul A evită utilizarea modului de lucru relocatabil. Aproximarea modulară are următoarele avantaje :

- Dezvoltare de program simplificată (In general e mai uşor să codezi,testezi şi depanezi mai multe programe simple decît unul complicat).

- Uşurează programarea (Programul B poate fi elaborat independent de mai mulţi programatori, pe cînd programul A pro - babil va trebui să fie făcut de un singur programator).

- Uşurează testarea (Programul B poate testa şi depana majoritatea modulelor imediat după asamblare. Programatorul A trebuie să-şi testeze programul ca pe un întreg. Programatorul B

mai are avantaj : dacă dezvoltarea senzorilor se face în ace - laşi timp cu programul. Dacă unul din aceşti senzori nu este gata, programatorul B poate continua dezvoltarea programului pentru senzorii care sînt gata. Pentru că programatorul A nu-şi poate testa programul pînă cînd nu sînt toţi senzorii gata, planul lui de testare depinde de cauze independente de el).

- Modificări în program (Dacă o schimbare a unui sen - zor necesită o schimbare de program, programatorul A trebuie să caute în întregul său program să găsească şi să schimbe codarea pentru acest senzor. Deci el va trebui să retesteze în - tregul program pentru a fi sigur că aceste schimbări nu au a-fectat pe nici unul din ceilalţi senzori. Prin contrast, pro - gramatorul B trebuie să se ocupe numai de modulul acestui unic senzor.

Pseudoinstrucţiuni utilizate pentru relocatare

Controlul numărătorului de adresă (Programe relocatabile)

Programele sau modulele relocatabile pot uriliza trei numărătoare de adresă. Pseudoinstrucţiunea ASEG, DSEG şi CSEG specifică care numărător de adrese e folosit.

ASEG specifică un segment de cod absolut. Chiar într-un modul relocatabil, putem asigna anumite secvenţe de cod la adrese specifice. De exemplu, rutina de restart apelată de RST necesită adresă specifică.

CSEG specifică un segment de cod relocatabil. In general numărătorul de adrese al CSEG e folosit pentru porţiuni de program care sînt conţinute în anumite forme de ROM, ca instrucţii în cod maşină, constante.

Numărătorul de adresă al DSEG specifică un segment de date relocabil, acest număr e folosit pentru elemente de program ce trebuie locatate în RAM.

Aceste pseudoinstrucţiuni facilitează controlul segmentării programului pe timpul asamblării. Programul LOCATE descris în ISIS-II System User's Guide, conţine posibilitatea controlului locatării segmentelor de program.

Indiferent de cîte ori apar ASEG, DSEG şi CSEG în program, asamblorul produce un singur modul continuu. Acest modul conţine patru segmente : codul, date, stivă şi memorie. Programele LINK şi LOCATE sînt folosite la combinarea segmentelor din

module diferite şi la relocatarea lor în memorie.

Pseudoinstrucţiunea ASEG

ASEG indică asamblorului să utilizeze numărătorul de adrese pentru segmentul absolut de program (absolut program segment).

Etichetă	Mnemonic	Operand
opţional:	ASEG	-

Nu se permite operanzi cu ASEG.

Toate instrucţiunile şi datele ce urmează după ASEG sînt asamblate în mod absolut. ASEG are efect pînă cînd apare un CSEG sau un DSEG.

Numărătorul de adrese al ASEG are valoare iniţială zero. ORG poate fi utilizat pentru asigurarea unei noi valori numărătorului de adrese al ASEG.

Cînd începe asamblarea, asamblorul presupune pseudoinstrucţiunea ASEG în acţiune. Deci un CSEG sau DSEG trebuie să preceadă prima instrucţiune sau prima definire de date într-un modul relocatabil. Dacă nu apare nici una dintre aceste pseudoinstrucţiuni în program, întregul program e asamblat în mod absolut şi poate fi executat imediat după asamblare fără utilizarea programelor LINK sau LOCATE.

Pseudoinstrucţiunea CSEG

CSEG indică asamblorului să asambleze instrucţiunile şi datele următoare (subsecvente) în mod relocatabil folosind numărătorul de adrese al segmentului de cod.

Etichetă	Mnemonic	Operand
opţional:	CSEG	{blank / PAGE / INPAGE}

Cînd un program conţine mai multe CSEG, toate CSEG din program trebuie să specifice acelaşi operand. Operandul unui CSEG nu are efect în asamblarea curentă dar este stocat cu codul obiect spre a fi trecut în programele LINK şi LOCATE. Programul LOCATE foloseşte această informaţie pentru determinarea limitelor relocatării cînd se alătură acest segment de

cod cu alte segmente de cod din alte programe. Semnificaţia operanzilor este următoarea :

- blank - acest segment de cod poate fi relocatat la limita următorului byte disponibil
- PAGE - acest segment de cod trebuie să înceapă la o limită de pagină cînd este relocatat. Limitele paginilor apar la multipli lui 256 de tyte-ţi începînd cu zero (0,256,512 etc.)
- IMPAGE- acest segment de cod trebuie să fie fixat pe o singură pagină cînd este relocatat.

CSEG are efect pînă la apariţia unui ASEG sau DSEG.

Numărătorul de adresă al segmentului de cod are iniţial valoarea zero .. ORG se poate utiliza pentru asigurarea unei noi valori numărotorului de adrese al CSEG.

Pseudoinstrucţiunea DSEG

DSEG indică asamblorului să asambleze instrucţiuni şi date subsecvente în mod relocabil folosind numărătorul de adrese al segmentului de date.

Etichetă	Mnemonic	Operand
opţional:	DSEG	{ blank PAGE IMPAGE }

Cînd apar mai multe DSEG în program, ele trebuie să specifice acelaşi operand de-a lungul programului. Operanzii pentru DSEG au aceeaşi semnificaţie ca cei pentru CSEG cu deosebirea că se aplică segmentului de date.

Nu există interacţiune între operanzii specificaţi pentru DSEG şi CSEG.

Deci un segment de cod poate fi relocatabil pe byte-uri în timp ce un segment de date e relocatabil pe pagini.

DSEG are efect pînă la întîlnirea unui ASEG sau CSEG.

Numărătorul de adrese al segmentului de date are valoarea iniţială zero.

ORG se poate utiliza pentru asignarea unei noi valori numărătorului de adrese al DSEG.

Pseudoinstrucţiunea ORG (mod de lucru relocatabil)

ORG poate fi utilizat pentru modificarea valorii numărătorului de adrese utilizat în momentul respectiv.

Etichetă	Mnemonic	Operand
opţional :	ORG	expresie

Sînt trei numărătare de adresă, dar numai unul din acestea este folosit la un moment dat într-un anumit punct din program. Aceasta depinde de una dintre ASEG, CSEG.

DSEG care este în acţiune.

Fiecare simbol din expresia operanzilor trebuie să fi fost anterior definit.O omisiune cauzează erori de fază pentru toate etichetele ce urmează după ORG şi o eroare de etichetă dacă eroarea nedefinită e ulterior definită.

Cînd ORG apare într-un segment de program relocatabil valoarea expresiei operandului său trebuie să fie sau absolută sau relocatabilă în cursul segmentului curent. Deci, dacă ORG apare în cursul unui segment de date, valoarea expresiei sale trebuie să fie relocatabilă în cursul segmentului de date. Apare o eroare dacă expresia se evaluează la o adresă în segmentul de cod.

Dacă o etichetă opţională e prezentă se asignează valoarea curentă numărătorului de adrese în acţiune înainte de ORG să fie fost executat.

Pseudoinstrucţiunea de linkare a programului (Program Linlage)

Programarea modulară şi posibilitatea de readresare face posibilă asamblarea şi testarea unor programe separate care trebuiesc reunite şi se execută ca un singur program. Uneori e necesar ca aceste programe separate să comunice informaţii între ele. Realizarea acestor comunicaţii se face cu pseudoinstrucţiunea de linkare a programului.

Un program poate împărţi adresele sale de date şi de instrucţii cu alte programe. Numai unităţile de informaţie avînd intrare în tabela de simboluri pot fi împărţite cu alte programe ; astfel, unităţilor de informaţie trebuie să li se asigneze un numr sau o etichetă cînd sînt definite în program. Unităţile de informaţie ce trebuie împărţite cu alte programe trebuiesc declarate în pseudoinstrucţiunea PUBLIC.

Programul dvs. poate avea acces direct la date sau instrucţiuni definite în alt program dacă cunoaşteţi actuala adresă a unităţii de informaţie,dar acest lucru nu este posibil cînd ambele programe folosesc relocatarea. Programul dvs. poate cîştiga acces la date sau instrucţiuni declarate ca PUBLIC în alte programe. De reţinut : asamblorul în mod normal indică eroare pentru orice referire la un nume sau etichetă dacă acestea nu au fost definite în program. Pentru a evita aceasta, trebuie să echipaţi asamblorul cu o listă de unităţi de informaţie în programul dvs. dar definite în alt program. Aceste unităţi de informaţie trebuie declarate cu EXTRN.

Pseudoinstrucţiunea PUBLIC

PUBLIC face fiecare simbol listat în cîmpul operand accesibil pentru alte programe.

Etichetă	Mnemonic	Operanzi
opţional:	PUBLIC	nume-listă

Fiecare unitate de informaţie în operandul nume-listă trebuie să fie nume sau etichetă asignată la date sau la o instrucţiune din alt loc din acest program. Cînd apar mai multe nume în listă ele trebuie separate prin virgulă.

Fiecare nume poate fi declarat PUBLIC numai odată într-un modul de program.

Cuvintele rezervate şi simbolul extern (vezi pseudoinstrucţiunea EXTERN mai jos) nu pot fi declarate ca simboluri PUBLIC.

PUBLIC poate apărea oriunde într-un modul de program. Dacă o unitate de informaţie în operandul nume-listă nu are corespondent de intrare în tabelul simbolurilor (deci nu e definită) e semnalată ca o eroare.

Exemplu :

PUBLIC   SIN, COS, TAN, SQRT

Pseudoinstrucţiunea EXTRN

EXTRN furnizează asamblorului o listă de simboluri la care se fac referiri în acest program dar definite în alt program.

Din această cauză asamblorul realizează legături cu celălalt program şi nu semnalează erori de referiri nedefinite.

Etichetă	Mnemonic	Operand
opţional:	EXTRN	nume-listă

Fiecare unitate de informare în lista de nume identifică un simbol care poate fi referit în acest program dar e definit în alt program. Cînd apar mai multe unităţi de informare în listă ele se despart prin virgulă.

Dacă un simbol în lista de nume a operandului e de asemeni definit în acest program de către utilizator sau e un simbol rezervat, efectul e acelaşi ca şi definirea aceuiaşi simbol de mai mult de odată în program.

Asamblorul semnalizează această eroare.

EXTRN poate apărea oriunde în decursul unui modul de program. Un simbol poate fi declarat EXTRN numai odată într-un modul de program. Simbolurile declarate PUBLIC numai pot fi declarate ca simboluri EXTRN.

Dacă se omite un simbol de pe lista de nume dar se face referire la el în program, simbolul e nedefinit. Asamblorul semnalizează această eroare. Se pot include simboluri în lista de nume a operandului la care nu s-au făcut referiri în program, fără a cauza erori.

Exemplu:

```
EXTRN      ENTRY, ADDRTN, BEGIN
```

<u>Pseudoinstrucţiunea NAME</u>

NAME asignează un nume modulului obiect generat de această asamblare.

Etichetă	Mnemonic	Operand
opţional:	NAME	numele modulului

NAME necesită prezenţa numelui modulului în cîmpul operand. Aceste nume trebuie să se conformeze legilor definirii simbolurilor.

Numele modulurilor sînt necesare ca să vă puteţi referi la un modul, şi să specificaţi secvenţele proprii ale modulelor cînd mai multe module sînt adiacente.

NAME trebuie să preceadă prima dată sau instrucţiune codată în programul sursă dar poate fi urmată de linii de comentarii şi control.

Dacă NAME lipseşte din program, asamblorul introduce o pseudoinstrucţiune NAME defectuoasă prin numele de modul MODULE. Acesta cauzează o eroare dacă încercaţi să legaţi împreună mai multe module MODULE. De asemeni, dacă faceţi o eroare codînd pseudoinstrucţiunea NAME, numele defectuos MODULE va fi asignat.

Numele de modul asignat prin NAME apare ca un cap de pagină în lista asamblată .

Exemplu :

NAME MAIN

Pseudoinstrucţiunea STKLN.

Indiferent de numărul modulelor programului obiect, legate împreună, se generează o singură stivă. STKLN facilitează specificare numărului de byte-uri necesare pentru stivă, pentru fiecare modul.

Etichetă	Mnemonic	Operand
opţional:	STKLN	expresie

Expresia operand trebuie evaluată la un număr ca dimensiune maximă pentru stivă.

Cînd un STKLN e omis, asamblorul produce un semnal de zero pentru STKLN lipsă. Acesta e folositor cînd mai multe programe sînt legate împreună ; numai o singură stivă va fi generată, deci numai un modul de program trebuie să specifice dimensiunea stivei. Totuşi puteţi furniza un STKLN dacă modulul trebuie testat separat şi utilizează stiva.

Dacă programul include mai mult de un STKLN, numai ultima valoare asignată e reţinută :

Exemplu :

STKLN 100

Cuvinte rezervate (STACK şi MEMORY)

STACK şi MEMORY nu sînt pseudoinstrucţiuni, dar prezintă interes pentru programele folosind relocatarea. Aceste cuvinte rezervate sîn referiri externe ale căror adrese sînt suplinite de programul LOCATE.

STACK e referirea simbolică la adresa de origine a stivei. Aveţi nevoie de această adresă pentru iniţializarea registrului Stack-Pointer. Puteţi astfel aşeza în stivă structuri de date pe această adresă folosind referiri simbolice ca STACK+1; STACK+2, etc.

MEMORY e referirea simbolică la primul byte al memoriei neutilizate după sfîrşitul programului. Din nou puteţi aşeza structuri de date la această adresă utilizînd referiri simbolice ca MEMORY+1, etc...

Sugestii pentru programare:

Testarea modulelor relocatabile

Abilitatea de a testa individual module de program e un avantaj major al programării modulare. Deoarece multe module de program din punct de vedere logic nu sînt destul de independente şi necesită o serie de modificări înainte de a fi testate.

Iniţializarea rutinelor

In multe programe complete, un număr de operaţii interne (ce nu fac parte din prelucrarea de date) sau proceduri de iniţializare sînt realizate chiar la începutul execuţiei. Dacă modulul de program pe care îl testaţi realizează o procedură de iniţializare asignată unui modul diferit, trebuie să repetaţi procedura în modulul ce urmează a fi testat. (De reţinut: totuşi se poate lega orice număr de module împreună, în vederea testării).

Unul din procedurile de iniţializare cele mai importante e aşezarea Stack-Pointerului. Programul LOCATE determină originea stivei.

Programul dvs. trebuie să includă următoarea instrucţiune de iniţializare a Stack-Pointerului :

```
LXI   SP,STACK
```

Intrare/Ieşire

Cînd testăm module de program, e posibil ca unele proceduri de intrare şi ieşire să apară în alte module. Programul dumneavoastră trebuie să simuleze toate aceste proceduri pe care le efectuează.

Deoarece sistemul de dezvoltare are capacităţi RAM m mult mai mari decît necesită testarea modulului de program utilizatorul trebuie să fie capabil să simuleze ieşiri şi intrări de date direct în memorie. Programul LOCATE furnizează o adresă pentru cuvîntul rezervat MEMORY; acesta e adresa primului byte al memoriei neutilizate după terminarea programului. Puteţi avea acces la această memorie folosind referinţele simbolice MEMORY; MEMORY+1 etc. Această memorie poate fi utilizată pentru stocarea datelor de test sau chiar pentru un program ce generează date de test.

Indepărtarea codurilor folosite pentru testare

După testarea programului asiguraţi-vă că a-ţi şters toate codurile inserate pentru testare. In particulat fiţi siguri că numai un modul din programul complet iniţializează stack-pointer-ul.

5. MACRO INSTRUCTIUNI ("MACROS")

Introducere

O macroinstrucţiune este o facilitate de-a înlocui un set de parametri cu un alt set. In timpul realizării programului, veţi găsi frecvent că multe secvenţe de instrucţii se repetă de mai multe ori cu numai cîţiva parametri schimbaţi.

Ca un exemplu, presupunem că se codează o rutină pentru mutarea a cinci byte-ţi dintr-o locaţie de memorie în alta. Puţin mai tîrziu vă veţi găsi în situaţia de-a coda o altă rutină pentru mutarea a patri byte-ţi dintr-un cîmp e sursă diferit într-un alt cîmp. Dacă cele două rutine folosesc acee - aşi tehnică de codare, veţi găsi că sînt identice exceptînd 3 parametrii : numărătorul de caractere, adresa de start a surseu şi adresa de start a destinaţiei. Desigur va fi mai uşor dacă există o cale de a regenera rutina originală, substituind noii parametrii fără a rescrie codul. Avantajele macro instrucţiunii sînt :

- Neplăcerea de a rescrie acelaşi set de instrucţiuni (şi probitatea erorilor) e redusă.
- Simbolurile utilizate în macro instrucţiuni por fi diferite încît să aibă înţeles numai în cadrul macro instrucţiunii. Deci codînd un program nu trebuie să vă temeţi că veţi duplica în mod accidental un simbol folosit în macro. De asemenea, o macroinstrucţiunea poate fi folosită ori de cîte ori în acelaşi program, este necesar, fără a duplica vreunul din simbolurile sale proprii.
- O eroare detectată în macro necesită o singură corectare fără a ţine cont de cîte ori se repetă macroinstrucţiunea în program. Aceasta reduce timpul de depanare.
- Efortul programării poate fi redus. Funcţiile utile pot fi adunate într-o bibliotecă şi recopiate pentru alte programe.

In plus macroinstrucţiunile pot fi folosite pentru a simplifica citirea programelor şi pentru programare structurată. Folosind microinstrucţiuni la blocurile segmentelor de cod, rezultă notaţii clare de program şi simplifică urmărirea cursului programului.

Ce este o macro instrucţiune

O "macro" poate fi descrisă ca o rutină definită într-o secvenţă formală de instrucţii prototip, care sînt chemate în cursul programului, au ca rezultat înlocuirea din aceste chemări cu expansiune de cod care constă în instrucţiunile corespunzătoare reprezentate.

Conceptul de macro definire, chemare şi expansiune poate fi ilustrat printr-o scrisoare tipică de afaceri, instrucţiunea prototip este textul de introducere. De exemplu putem defini o "macro" CNFIRM cu textul:

"Air Flight vă spune bun venit ca pasager".

Numărul dvs. de zbor FNO decolează la DTIME şi aterizează în DEST LA ATIME.

Aceste "macro" au 4 parametrii fictivi ce trebuie înlocuiţi, cînd macro e apelată prin: numărul zborului actual, timpul plecării, destinaţia, timpul sosirii. Apelarea la "macro" poate arăta în felul următor :

CNFIRM 123,'10:45','Ontario','11:52'

O a doua "macro", CAR, poate fi aplicată dacă pasagerul a cerut ca o maşină de închiriat să fie rezervată la aeropotrul de destinaţie.

Această "macro" poate avea textul:

Rezervarea dvs. de automobil a fost confirmată cu agenţia de închiriat maşini MAKE.

In final, o "macro" GREET poate fi definită pentru specificarea numelui pasagerului:

Dear NAME:

Textul întreg al scrisorii de afaceri (fila sursă) va fi:

```
GREET 'Ms.Scannel'
CNFIRM 123,'10:45','Ontario',11:52'
CAR 'Blotz'
```

Sperăm să aveţi o călătorie plăcută.

Cu stimă.

Cond fila sursă e trecută printr-un procesor de "macro", apelările la "macro" vor produce scrisoarea următoare :

Dear Ms.Scannael :

Air Flight vă spune bun venit ca pasager. Numărul dv. de zgor 123 decolează la 10:45 şi aterizează la 11:52 . Rezervarea dv. de automobil a fost confirmată de agenţia Blotz.

Sperăm să aveţi o călătorie plăcută.

Cu stimă.

In timp ce exemplul ilustrează substituirea parametrilor într-o "macro" dezvăluie în acelaşi timp legătura dintre expansiunea macro şi asamblor.

Scopul procesorului de macro e să genereze un program sursă care urmează să fie asamblat.

Macro în funcţie de subrutine

La acest punct se tratează diferenţele dintre "macro" şi subrutinele apelate prin CALL. Amondăuă ajută structurarea progrsmului şi reduc codarea rutinelor executate frecvent.

O diferenţă între cele două e că : subrutinele necesită ramificaţie la o altă parte a programului, pe cînd "macro" sînt generate "în linie".

Deci, programul conţine numai o versiune a subrutinei date, dar conţine atîtea versiuni ale unei "macro" date, cîte sînt apelate de această /macro".

De reţinut că accentul pus pe "versiuni" în proporţiile anterioare este o diferenţă majoră între "macro" şi subrutine. O "macro" nu trebuie în mod necesar să genereze acelaşi cod de cîte ori este apelată. Schimbînd parametrii în apelarea la "macro", puteţi schimba codul sursă pe care îl generează "macro". In plus, parametrii macro pot fi testaţi în timpul asamblării de pseudoinstrucţiuni de asamblare condiţională.Aceste două dispozitive permit definirea unui "macro" de uz general pentru a genera un cod sursă de largă utilizare pentru situaţii de programare particulare.

De reţinut că expansiunea "macro" şi generalizarea oricărui cod au loc în timpul asamblării şi la nivelul codului sursă. In contrast o subrutină generalizată rezidă în programul

dv. şi necesită timp de execuţie.

Se poate întîmpla să obţinem rezultate similare folosind fie o "macro", fie o subrutină. Determinarea variantelor optime nu poate fi întodeauna univocă. In multe cazuri, folo - sind o singură subrutină în loc de mai multe "macro" "în linie" se poate reduce dimensiunea totală a programului. In situaţii care implică un număr mare de parametrii, folosirea de "macro" poate fi mai eficientă. De asemenea e de reţinut că "macro" poate apela la subrutine şi subrutinele pot conţine "macro".

Folosirea "macro"-urilor

Asamblorul recunoaşte următoarele operaţiuni "macro":

- pseudoinstrucţiunea MACRO
- pseudoinstrucţiunea ENDM
- pseudoinstrucţiunea LOCAL
- pseudoinstrucţiunea REPT
- pseudoinstrucţiunea IRP
- pseudoinstrucţiunea IRPC
- pseudoinstrucţiunea EXTIM
- Apelarea la "macro"

Toate pseudoinstrucţiunile listate deasupra sînt de - crise la macro definiţii.

Definiţii macro

"Macro"-urile trebuie definite în program înainte de a fi utilizate. O definire macro e iniţializată de pseudoinstrucţiunea MACRO a asamblorului, care listează numele prin care "macro" poate fi mai tîrziu apelat şi care listează parametrii fictivi ce urmează a fi înlocuiţi în timpul expansiunii macro. Definirea "macro"-ului se termină prin ENDM. Instrucţiunile prototip adiacente lui MACRO şi ENDM poartă denumirea de corpul macro.

Cînd etichetele simbol folosite în corpul macro au scop "global", rezultă o multitudine de erori de definire de simbol dacă "macro" e chemată mai mult de o dată. Unei etichete i se poate atribui un scop local prin LOCAL. Această pseudoinstrucţiune asignează o valoare unică simbolului ori de cîte ori "macro"-ul a apelat şi expandat. Parametrii fictivi au deasemenea scop limitat.

Ocazional puteţi dori să duplicaţi un bloc de cod de

mai multe ori, fie într-un "macro" fie în linie cu un alt cod sursă. Acest lucru poate fi realizat cu un efort de codare minim, folosind REPT (repetă blocul) IRP (repetare nedefinită), şi IRPC (repetare nedefinită de caracter). Ca şi MACRO, aceste pseudoinstrucţiuni se termină prin ENDM.

EXIT face posibilă ieşirea alternată din "macro". Cînd se înlocuieşte EXTIM aceasta termină "macro" în curs întocmai ca şi cînd ar fi fost întîlnită ENDM.

Presudoinstrucţiunea de definire macro

Pseudoinstrucţiunea MACRO

Etichetă	Mnemonic	Operand
nume	MACRO	Parametrii(i) fictiv(i) operaţional(i)

Numele din cîmpul etichetă specifică numele corpului macro ce a fost definit. Orice nume simbolic valid definit de utilizator şi poate fi folosit ca nume de "macro". De reţinut că acest nume trebuie să fie prezent şi să nu se termine prin două puncte. Un parametru fictiv poate fi orice nume simbolic valid definit de utilizator sau poate fi nul. Cînd se listează mai mulţi parametrii ei trebuie separaţi prin virgule, Scopul unui parametru fictiv e limitat la definirea sa specifică "macro". Dacă un simbol rezervat e utilizat ca parametru fictiv, valoarea se rezervată nu e recunoscută. De exemplu :dacă codaţi A,B,C drept listă de parametrii fictivi, substituirea are loc normal. Totuşi nu puteţi utiliza acumulatorul sau registrele B şi C în "macro". Din cauza ariei de acţiune limitată a parametrilor fictivi, utilizarea acestor registre nu e afectată în afara definiţiei "macri"-urilor.

Parametrii fictivi nu sînt recunoscuţi în comentariu. Nu are loc substituire pentru aceşti parametrii.

Parametrii fictivi pot să apară într-un şir de caractere. Deci, parametrii fictivi trebuie să fie adiacenţi unui caracter de legătură (&).

Orice instrucţie maşină sau pseudoinstrucţiune de asamblare aplicabilă poate fi inclusă în corpul macro. Caracteristica de distincţie a textului macro prototip este faptul că părţi din el pot fi făcute variabile prin aşezarea parametri-

lor fictivi substituabili în cîmpurile de instrucţiune.Aceşti parametrii fictivi sînt aceeaşi ca simbolurile în cîmpul operand al pseudoinstrucţiunii MACRO.

Exemplu"

Definirea macro MAC1 cu parametrii fictivi G1,G2 şi G3.

Observaţi : următoarea macro definiţie conţine o eroare potenţială care se va clarifica în descrierea pseudoinstrucţiunii LOCAL.

```
MCA1     Macro   G1,G2,G3   ;pseudoinstrucţiunea MACRO
MOVES:   LHLD    G1          corpul macro
         MOV     A,M
         LHLD    G2
         MOV     B,M
         LHLD    G3
         MOV     C,M
         ENDM               ;pseudoinstrucţiunea ENDM
```

Pseudoinstrucţiunea ENDM

Etichetă	Mnemonic	Operand
—	ENDM	- -

ENDM e chemat să termine o definiţie macro şi urmează după ultima instrucţiune prototip. E de asemenea chemat să termine blocurile de repetare a codului,definite de REPT, IRP şi IRPC.

Orice dată ce apare în cîmpul de etichetă sau operand al lui ENDM cauzează eroare.

Observaţii : Deoarece apelările macro încorporate nu sînt expandate în timpul definirii macro, ENDM care trebuie să termine o apelare macro exterioară nu poate fi conţinută într-o apelare tabelată interior. (Vezi Definiţii macro încorporate tabelate mai departe în acest capitol).

Pseudoinstrucţiunea LOCAL

Etichetă	Mnemonic	Operand
- -	LOCAL	nume de etichetă

Numele de etichetă specificate sînt definite a avea înţeles numai în expansiunea macro în curs. Ori de cîte ori

"macro" e apelată şi expandată, asamblorul asignează fiecărui simbol local un simbol unic de forma ??nnnn. Asamblorul asignează ??0001 primului simbol local,??0002 celui de al doilea etc.... Cel mai recent nume simbolic generat întodeauna indică numărul total de simboluri create pentru toate expansiunile macro, Asamblorul nu dublează niciodată aceste simboluri. Utilizatorul trebuie să evite codarea simbolurilor în forma ??nnnn, pentru evitarea conflictului cu simbolurile generate de asamblor.

Parametrii fictivi incluşi într-o apelare la macro nu pot fi operanzi ai unei pseudoinstrucţiuni. LOCAL. Aria de acţiune a unui parametru fictiv se reduce întotdeauna la propria sa definiţie macro.

Simbolurile locale pot fi definite numai în cadrul definiţiei macro. Orice număr de pseudoinstrucţiune LOCAL pot apare într-o definiţie macro, dar ele trebuie să urmeze după apelarea macro şi trebuie să preceadă prima linie a codului prototip.

Deci un LOCAL care apare în afara unei definiţii macro cauzează o eroare. De asemenea un nume apărînd.în cîmpul de etichetă al lui LOCAL cauzează o eroare.

Exemplu:

Definirea lui MAC1 (folosit ca un exemplu în descrierea lui MACRO) conţine o eroare potenţială deoarece simbolul MOVES nu a fost declarat local. Aceasta e o eroare potenţială atîta timp cît MAC1 a apelat o singură dată în program şi programul însuşi nu foloseşte MOVES ca simbol.

Dacă MAC1 e apelat mai mult de o dată, sau dacă program utilizează simbolul MOVES, MOVES va fi un simbol multiplu definit. Această eroare potenţială poate fi evitată prin numirea lui MOVES în cîmpul operand al unei "LOCAL".

```
MAC1     MACRO    G1,G2,G3
         LOCAL    MOVES
MOVES    LHLD     G1
         MOV      A,M
         LHLD     G2
         MOV      B,M
         LHLD     G3
         MOV      C,M
         ENDM
```

Presupunem că MAC1 e singura "macro" în program şi e apelată de două ori. Prima dată MAC1 e expandat . MOVES e în - locuit cu simbolul ??0001 ; a doua oară MOVES a înlocuit cu ??0002. Deoarece asamblorul întîlneşte numai aceste simboluri speciale de înlocuire, programul poate conţine simbolul MOVES fără a cauza definirii multiple.

Pseudoinstrucţiunea REPT

Etichetă	Mnemonic	Operand
opţional :	REPT	expresie

REPT provoacă o secvenţă de linii de cod sursă spre a fi repetate de "expresie" ORI. Toate liniile ce apar între REPET şi ENDM subsecvent constituie blocul ce urmează a fi repetat.

Cînd "expresie: conţine nume simbolice, asamblorul trebuie să întîlnească definiţia simbolului, înainte de a întîlni expresia.

Inserţia blocului de repetat se face în linie cînd asamblorul întîlneşte REPT. Nu e necesară o apelare explicită pentru a provoca inserţie de cod, deoarece definiţia e o apelare implicită pentru expansiune.

Exemplu : 1 :

Rotarea acumulatorului de 6 ori

```
ROTR 6 :    REPT     6
            RRC
            ENDM
```

Exemplu 2 :

REPT următoare generează un cod sursă pentru o rutină ce umple un cîmp de 5 byte cu caracterul stocat în acumulator:

```
CODUL PROGRAMULUI         CODUL GENERAT
LHLD      CNTR1           LHLD      CNTR1
REPT 5    5               MOV       M,A
MOV       M,A             INX       H
INX       H               MOV       M,A
ENDM                      INX       H
                          MOV       M,A
                          INX       H
                          MOV       M,A
                          INX       H
                          MOV       M,A
                          INX       H
```

Exemplu 3 :

Ilustrăm folosirea lui REPT pentru a genera o rutină de multiplicare, Multiplicarea se realizează printr-o serie de deplasări. Dacă această tehnică nu e familiară, vezi exemplul din cap.6. Exemplu din cap.6 foloseşte o buclă de program pentru multiplicare. Exemplu de faţă înlocuieşte bucla cu 7 repetiţii ale celor 4 instrucţii încadrate de REPT - ENDM.

Reţineţi : expansiunea specificată de REPT provoacă generarea etichetei SKIPAD de 7 ori. Deci, SKIPAD trebuie declarat local în această "macro".

```
FSTMUL :  MVI    D,0      ;rutină de multiplicare rapidă
          LXI    H,0      ;multiplică E*A - retultat de 16 bit
                          ;în H&L
          REPT   7
          LOCAL  SKIPAD
          RLC             ;;următorul  bit de multiplicare
          JNC    SKIPAD   ;;nu adună dacă BIT=0
          DAD    D        ;;adună multiplicandul în răspuns
SKIPAD:   DAD    H
          ENDM
          RLC
          RNC
          DAD    D
          RET
```

Acest exemplu ilustrează o modalitate clasică de programare; viteză în funcţie de memorie . Cu toate că acest exemplu execută mult mai repede de cît exemplul din cap.6, el necesită mai multă memorie.

<u>Pseudoinstrucţiunea IRP</u>

Etichetă	Mnemonic	Operand
opţional:	IRP	parametru fictiv, listă

Cîmpul operand al lui IRP (repetare nedefinită) trebuie să conţină un parametru fictiv macro urmat de o listă a parametrilor reali încadrată între paranteze ascuţite. IRP expandează codul prototip macro asociat lui, substituind primul parametru

real pentru fiecare întîlnire a parametrului fictiv. IRP expandează apoi codul prototip, substituind din nou al doilea parametru real al listei. Procesul continuă pînă la epuizarea listei.

Lista parametrilor reali ce urmează a fi înlocuiţi de parametrul fictiv, trebuie să fie inclusă în paranteze ascuţite, Unităţile individuale de informaţie din listă trebuie separate prin virgule. Numărul parametrilor actuali din listă controlează numărul de repetări ale corpului "macro".O listă cu n unităţi de informaţie provoacă n repetiţii, O listă goală (fără parametri codaţi) specifică o listă de operand nulă. IRP generează o copie a corpului macro substituind un zero la fiecare întîlnire a parametrului fictiv. De asemenea, două virgule fără caractere între ele generează un parametru nul în listă.

Exemplu :

Următoarea secvenţă de cod adună byte-urile datelor din diferite zone ale memoriei şi le stochează apoi în byte-uri consecutive începînd cu adresa lui STORIT :

CODUL PROGRAMULUI		CODUL GENERAL	
LXI	H,STORIT	LXI	H,STORIT
IRP	X, FLD1,3E20H,FLD3	LDA	FLD1
LDA	X	MOV	M,A
MOV	M,A	INX	H
INX	H	LDA	3E20H
ENDM		MOV	M,A
		INX	H
		LDA	FLD3
		MOV	M,A
		INX	H

Pseudoinstrucţiunea IRPC

Etichetă	Mnemonic	Operand
opţional :	IRPC	patametru fictiv, text

IRPC (repetare nedefinită de caracter) provoacă o secvenţă de instrucţiuhi prototip macro, ce trebuie repetate pentru fiecare caracter de text al parametrului real specificat. Dacă rîndul de text e inclus în paranteze ascuţite opţionale, orice delimitări ce apar în rîndul de text sînt tratate simplu

ca text ce urmează a fi substituit în cadrul prototip.

Asamblorul generează o iterație a codului prototip pentru fiecare caracter în rîndul de text. Pentru fiecare iterație asamblorul înlocuiește următorul caracter din rînd la fiecare întîlnire a parametrului fictiv. O listă de n caractere de text generează n repetări ale corpului macro al lui IRPC. Un rînd gol specifică un operand real nul. IRPC generează o copie a corpului macro substituind un nul pentru fiecare întîlnire a parametrului fictiv:

	CODUL PROGRAMULUI		CODUL GENERAT	
	LHLD	DATE-1	LHLD	DATE-1
MVDATE:	IRPC	X,1977	INX	H
	INX	H	MVI	M,1
	MVI	M,X	INX	H
	ENDM		MVI	M,9
			INX	H
			MVI	M,7
			INX	H
			MVI	M,7

IRPC oferă posibilitatea de a fi tratat fiecare caracter din rînd în mod individual : înlănțuirea (descrisă mai tîrziu) asigură capacitatea de a construi rînduri de text din caractere individuale.

<u>Pseudoinstrucțiunea EXITM</u>

Eticheta	Mnemonic	Operand
opțional:	EXITM	- -

EXITM asigură o metodă alternată de terminare a expansiunii macro sau a repetiției secvențelor de cod ale lui REPT, IRP și IRPC. Cînd EXITM e întîlnit, asamblorul ignoră toate instrucțiunile prototip macro, localizate între EXITM și ENDM corespunzător . Rețineți că, EXITM poate fi utilizat împreună cu ENDM dar nu în locul lui ENDM. Cînd se utilizează o "macro" încorporată (nested), EXITM provoacă o ieșire la nivelul anterior al expansiunii macro. O EXITM în REPT, IRP sau IRPC termină nu numai expansiunea curentă , dar și toate iterațiile subsecvente. Orice date ce apar în cîmpul operand al EXITM cau-

zează eroare.

Exemplu :

EXITM se utilizează în mod tipit la suprimarea expansiunii macro nedorite, In exemplul următor expansiunea macro e terminată cînd EXITM e asamblat, deoarece condiţia X EQ 0 e adevărată.

```
MAC3    MACRO       X,V
        .
        .
        .
        IF X EQ 0
        EXITM
        .
        .
        .
        ENDM
```

<u>Operaţori macro speciali</u>

In anumite cazuri speciale, regulile normale de operare cu "macro"-uri nu dau rezultate.

Presupunem, de exemplu că vrem să specificăm trei parametri reali şi al doilea paramteru se întîmplă să fie un caracter virgulă. Pentru asamblor lista PARM1,,, PARM3 pare a fi o listă de 4 parametri, dintre care cel de al doilea şi al treilea lipsesc. Lista poate fi parcursă corect incluzînd virgula între paranteze ascuţite. : PARM3. Aceşti operatori speciali arată asamblorului să accepte caracterul încadrat (virgula) ca un parametru real şi nu o delimitare.

Asamblorul recunoaşte un număr de operatori care permit operaţii speciale :

& Semn de legătură. Utilizat la legarea (LINK) de text şi de parametrii fictivi.

Paranteze ascuţite. Utilizate la delimitări de text, ca liste, care conţin alte delimitări, Reţineţi că blancurile sînt tratate ca delimitatori. Prin urmare, cînd un parametru actual conţine blancuri (trecînd instrucţia MOV A,M de exemplu) parametru trebuie inclus între paranteze ascuţite. Acest lucru e de asemenea adevărat pentru orice alt delimitator ce trebuie considerat ca parte a unui parametru real.

A trece peste un text ca de exemplu o apelare la macro încorporată foloseşte un set de paranteze ascuţite pentru fiecare nivel de incluziune.

Punct şi virgulă dublate. Utilizate în faţa unui comentariu într-o definiţie macro, pentru a preveni includerea comentariului în expresia "mcaro"-ului şi reduce necesităţile de stocare. Comentariul apare totuşi în lista definiţiilor.

Semnul exclamării (caracter de fugă). Plasat în faţa unui caracter (de obicei un delimitator), spre a fi interpretat ca un text literalizat într-un parametru real. Se utilizează mai ales pentru a interpreta paranteza ascuţită ca parte a unui parametru actual. Pentru a interpreta un semn de exclamare literalizat, editează II. Intoarcerile carului nu pot trece drept parametrii reali.
Semnul ! e întotdeauna rezervat în timpul construirii unui parametru real. Nu se repetă cînd un parametru real e înlocuit cu unul fictiv, exceptînd cazul cînd înlocuirea are loc pentru construirea unui alt parametru real.

In anumite cazuri nu e necesar să se asocieze un parametru la macro. E necesar totuşi, să se indice omiterea parametrului. Parametrul omis (sau nul) poate fi reprezentat de doi delimitatori consecutivi ca în lista ; PARM1,, PARM3. Un parametru nul poate fi deasemenea reprezentat prin două ghilimele consecutive :
",PARM2,PARM3. Reţineţi că un nul e diferit de un blanc : un blanc e un caracter ASCII cu reprezentarea hexa 20H ; un nul nu are reprezentare de caracter. In lista asamblată nulul arată la fel cu blancul, dar aceasta numai din cauză că nu a avut loc înlocuirea. Programatorul trebuie să decidă sensul parametrului nul. Deşi mecanismul e oarecum diferit, defectele luate în consideraţie pentru controalele asamblorului dau un bun exemplu a ceea ce poate însemna un parametru nul. De exemplu : codînd

MOD85 ca un control de asamblor specifică că asamblorul generează cod obiect pentru 8085. Absenţa acestui control (deci un parametru nul) specifică, că asamblorul generează programului obiect numai pentru 8080.

Exemplu :

Intr-o "macro" cu parametrii fictivi W,X,Y,Z e acceptabil ca ori X ori Y să fie nul, dar nu amîndouă. Următorul IF testează condiţia de eroare:

IF NUL X&Y
EXTIM

Cînd o "macro" e expandată, orice semn de legătură precedînd sau urmînd un parametru fictiv într-o definiţie macro , e repus şi înlocuirea parametrului real intervine în acest pct. Cînd nu e adiacent unui parametru fictiv, semnul de legătură nu e repus şi e luat ca parte al textului expansiunii macro.

Observaţi semnul de legătură trebuie să fie imediat adiacent textului de linkat ; blancurile care intervin nu sînt admise. Dacă definiţii macro tabelate conţin semne de legătură, singurele semne de legătură (înlăturate) sînt cele adiacente parametrilor fictivi ce aparţin definiţiei macro curente care a început să fie expandată. Toate semnalele de legătură tre - buiesc înlăturate pe timpul expansiunii corpului macro.

Excepţiile forţează eroare de caracter ilegal.

Semnele de legătură plasate în interiorul rîndurilor sînt recunoscute ca delimitări de înlănţuiri cînd sînt adia - cente parametrilor fictivi; în mod similat parametrii fictivi din rîndurile de caractere sînt recunoscuţi numai cînd sînt adiacenţi la semne de legătură. Semnele de legătură nu sînt recunoscute ca operatori în comentarii.

<u>Macrodefiniţii incorporate</u>

O definiţie macro poate fi conţinută în corpul unei alte definiţii macro(aceasta fiind definiţia incorporată) Corpul unei macro cuprinde toate textele (incluzînd definiţia macro incorporate) limitate de o pereche de pseudoinstrucţi - uni .MACRO şi ENDM. Asamblorul permite incorporarea unui număr

nelimitat de definiţii macro.

Cînd se apelează pentru expansiune o macro de nivel mai înalt, următoarea macro de nivel mai scăzut e definită şi pregătită pentru a fi apelată pentru expansiune. O macro de nivel mai scăzut nu poate fi apelată fără ca toate definiţiile macro de nivel mai înalt să fi fost apelate şi expandate.

O nouă macro poate fi definită sau o macro existentă poate fi redefinită printr-o definiţie macro încorporată. Acest lucru fiind posibil dacă numele macro-ului încorporat e o nouă etichetă ori a fost iniţial stabilit ca un parametru fictiv într-o definiţie macro de nivel mai înalt. Deci, întotdeauna, cînd o macro de nivel mai înalt e apelată, o macro de nivel inferior poate fi definită diferenţiat dacă cele două conţin parametrii fictivi comuni. Această redefinire poate fi constisitoare din punct de vedere al vitezei de execuţie al asamblorului.

Deoarece IRP, IRPC şi REPT sînt blocuri ce sontituie definiţia macro, ele de asemenea pot fi încorporate în alte definiţii create de IRP,IRPC REP şi MACRO. In plus un element în lista parametrilor reali din IRP sau IRPC (incluşi în paranteză ascuţită) poate fi el însăşi o listă de parametrii puşi în paramteză; listele de parametrii vor conţine elemente care sînt de asemenea liste.

Exemplu :

```
LISTS     MACRO        PARAM1 , PARAM2
          .
          .
          .
          ENDM
          .
          .
          .
          LISTS  A,  B,C
```

Apelări macro (MACRO CALLS)

Cînd o macro a fost definită, ea poate fi apelată ori de cîte ori este nevoie în timpul programului. Apelarea constă din numele macro-ului şi oricare parametru real care urmează să înlocuiască parametruu fictivi în timpul expansiunii macro. In timpul asamblării fiecare apelare macro e înlocuită de codul definiţiei macro;

parametrii fictivi sînt înlocuiţi de parametrii reali.

Format de apelare macro

Etichetă	Mnemonic	Operand
opţional:	nume macro	parametru(i) real(i) opţional(i)

Asamblorul trebuie să întîlnească definiţia macro înainte de prina apelare a acelui macro. Astfel apelarea macro e considerată a fi un mnemonic ilegal. Asamblorul inserează corpul macro identificat de numele macro ori de cîte ori întîlbeşte o apelare la o macro anterior definită în program.

Poziţionarea parametrilor reali în apelarea macro e critică pînă cînd înlocuirea parametrilor se bazează doar pe poziţie. Primul parametru real listat înlocuieşte la fiecare întîlnire pe primul parametru fictiv listat : al doilea parametru înlocuieşte al doilea parametru fictiv şi aşa mai departe. Cînd codăm o apelare macro, trebuie să fim siguri că listarea parametrilor reali se face în secvenţa potrivită pentru macro.

Reţineţi că blancurile sînt tratate ca delimitatori. Totuşi, cînd un parametru actual conţine blancuri(ajungînd la instrucţia MOV A,M de exemplu) parametrul trebuie inclus între paranteze ascuţite. Aceasta e de asemenea valabil pentru oricare alt delimitator care e interpretat ca parte a unui parametru real. Revenirea carului (carriage returns) nu poate di interpretată ca un parametru real.

Dacă o apelare macro specifică mai multi parametrii actuali decît sînt listaţi în definiţia, macro, parametrii în plus sînt ignoraţi. Dacă în apelare apar mai puţini parametrii decît în definiţie, un nul înlocuieşte fiecare parametru omis.

Exemplu :

Următorul exemplu arată două apelări pentru macro-ul LOAD. LOAD e definit ca mai jos.

```
LOAD      MACRO     G1,G2,G3
          LOCAL     MOVES
MOVES:    LHLD      G1
          MOV       A,M
          LHLD      G2
          MOV       B,M
          LHLD      G3
          MOV       C,M
          ENDM
```

LOAD încarcă acumulatorul cu un byte al datei la locaţia specificată de primul parametru real, registrul B cu byte din parametrul al II-lea şi registrul C cu un byte din parametrul al III-lea.

In primul moment LOAD e apelată, e utilizată ca o parte dintr-o rutină de inversare a ordinii a trei byte din memorie. A doua apelare la LOAD e o parte a unei rutine care adună conţinutul registruluiB la acumulator şi apoi compară rezultatul cu conţinutul registrului C.

```
    PROGRAM PRINCIPAL                    SUBSTITUTION

JNZ NEXT                                 JNZ NEXT
LOAD  FLD,FLD+1,FLD+2          ??0001:   LHLD FLD
MOV   M,A    ;BYTE-ţi inversaţi          MOV A,M
DCX   H                                  LHLD  FLD+1
MOV   M,B                                MOV   B,M
DCX   H                                  LHLD  FLD+2
MOV   M,C                                MOV   C,M
LOAD  3EOH,BYTE,CHECK(control)           MOV   M,A      ;BYTE-uri inversate
ADD   B    ;CHECH DIGIT                  DCX   H
CMP   C                                  MOV   M,B
CNZ   DGTBAD                             DCX   H
                                         MOV   M,C
                               ??0002:   LHLD  3EOH
                                         MOV   A,M
                                         LHLD  BYTE
                                         MOV   B,M
                                         LHLD  CHECK
                                         MOV   C,M
                                         ADD   B        ;CHECK DIGIT
                                         CMP   C
                                         CNZ   DGTBAD
```

Macro apelări încorporate

Apelările macro (incluzînd orice combinații de construcții IRP, IRPC și REPT încorporate) pot fi încorporate în definiția macro pînă la opt nivele. Macro început a fi apelată nu trebuie definită cînd macro care a inclus e definită ; totuși, trebuie definită înainte de a apela la macro-ul care o include.

O definiție macro poate de asemenea conține apelări la ea însăși încorporate în ea (apelări macro recursive) pînă la opt nivele atîta timp cît expansiunile macro recursive poate fi pînă la urmă terminate. Această operație poate fi controlată folosind pseudoinstrucțiuni de asamblare condiționată (IF,ELSE,ENDIF) .

Exemplu :

Avem o autoapelare macro în ea însăși cu cinci timpi după ce a fost apelată din altă parte a programului.

```
PARAM1      SET       5
RECALL      MACRO
            .
            .
            .
            IF        PARAM1 NE 0
PARAM1      SET       PARAM1 - 1
            RECALL                        ; RECURSIVE CALL
            ENDIF
            .
            .
            .
            ENDM
```

Expansiune macro

Cînd se apelează o macro, parametrii reali ce urmează a fi substituiți, în cadrul prototip pot fi trecuți (passed) în două moduri. In mod normal, înlocuirea parametrilor reali cu parametrii fictivi e o simplă substituție de text. Parametrii nu sînt evaluați pînă cînd macro nu e expandată.

Dacă semnul procent (%) precede un parametru real în apelarea macro, oricum parametru este evaluat imediat înainte de intervenirea expansiunii, și e considerat ca un număr zecimal reprezentînd caloarea parametrului. In cazul lui IRPC un % precedînd parametru real, provoacă tratarea întregului rînd de text ca un parametru singular. O iterație IRPC are loc pentri fiecare digit în rîndul zecimal interpretat ca rezultat al evaluării imediate a rîndului de text.

Mecanismul normal de interpretare a parametrilor reali e adecvat pentru majoritatea aolicaţiilor. Folosind semnul % pentru preevaluarea parametrilor, e necesară numai cînd valoarea parametrului e diferită în contextul local al definiţei macro dacă e comparat cu valoarea globală din afara definiţiei macro.

Exemplu :

Macro arătat în acest exemplu generează un număr de instrucţiuni de rotire (rotate instrucţions).Parametrii trecuţi (posed) în apelarea macro determină numărul poziţiilor acumulatorului care urmează să fie rotit şi dacă sînt generate instrucţii de rotire la dreapta sau la stînga. Cîteva apelări tipice pentru această macro sînt :

```
SHIFTR          'R',3
SHIFTR          L,%COUNT-1
```

A doua apelare macro prezintă o expresie folosită ca parametru. Această expresie trebuie să fie evaluată imediat şi nu interpretată ca text.

Definiţia lui SHIFTR e prezentată mai jos. Această macro foloseşte IF condiţional pentru a testa validitatea primului parametru. De asemenea, REPT e încorporată în SHIFTR.

```
SHIFTR     MACRO               X,Y
           IF X EQ  'R'
              REPT Y
                RAR
              ENDM
           ENDIF
           IF X NE 'L'
              EXITM
           ELSE
             REPT Y
               RAL
             ENDM
           ENDIF
           ENDM
```

Zigzagul care apare în definiţia lui SHIFTR ilustrează grafic relaţia dintre IF,ELSE,ENDIF şi REPT,ENDM. Asemenea reprezentare nu se cere în program dar poate fi dorită ca documentaţie.

SHIFTR nu generează nimic dacă primul parametru nu e R sau L.

De aceea apelările următoare nu produc cod. Rezultatul în cadrul obiect e ca şi cum SHIFTR nu ar apărea în programul sursă.

```
SHIFTE 5
SHIFTR 'B',2
```

Următoarea apelare la SHIFTR generează trei instrucţiuni RAR :

```
SHIFTR    'R',3
```

Presupunem că un SET în altă parte în programul sursă, a dat lui COUNT valoarea 6.Următoarea apelare generează 5 instrucţiuni RAL :

```
SHIFTR   'L',%COUNT-1
```

Urmează o redefinire a lui SHIFTR. In această definiţie, reţinei că. înlănţuirea e folosită la formarea codului de operare al lui RAR sau RAL. Dacă o apelare la SHIFTR specifică un alt caracter decît R sau L, se generează coduri de operare ilegale. Asamblorul semnalează toate codurile de operare ilegale ca erori.

```
SHIFTR      MACRO       X,Y
            REPT        Y
            RA&X
            ENDM
            ENDM
```

Macro-uri nule

O macro poate cuprinde numai pseudoinstrucţiuni MACRO şi ENDM. Deci, următoarea e o definiţie macro legală :

```
NADA     MACRO      P1,P2,P3,P4
         ENDM
```

O apelare la această macro nu produce cod surdă deci nu are efect în program.

Deşi nu are sens scriem un astfel de macro, corpul macro nul (sau gol) are o aplicaţie practică. De exemplu, toate instrucţiunule prototip macro pot fi închise între IF-ENDIF. Cînd nici una din condiţiile specificate nu e astisfăcută,tot ce rămîne din macro MACRI şi ENDM.

Exemple de macro

Următoarele mostre de macro vor demonstra folosirea pseudoinstrucţiunii şi operatorilor macro.

Exemplul 1 :

IRPC incorporat

Reţineţi : operandul al treilea al macro-ului exterior apare în rîndul de caractere pentru IRPC :

```
MOVE    MACRO       X,Y,Z
        IRPC        PRAM,Z
        LHLD        X&&PARAM
        SHLF        Y&&PARAM
        ENDM
        ENDM
```

Presupune că programul conţine apelarea MOVE SRC,DST, 123. Al treilea parametru al acestei apelări e trecut lui IRPC. Aceasta are acelaşi efect ca şi codarea lui IRPC PARAM,123. Cînd e expandat, MOVE generează următorul cod sursă :

```
LHLD    SRC1
SHLD    DST1
LHLD    SRC2
SHLD    DST2
LHLD    SRC3
SHLD    DST3
```

Reţineţi : folosirea înlănţuirii la formarea etichetelor în acest exemplu

Exemplu 2 :

Macro încorporate folosite la generearea pseudoinstrucţiunii DB.

Acest exemplu generează un număr de pseudoinstrucţiuni DN 0, fiecate cu et cheta sa proprie. Două macro sînt folosite în acest scop,: INC şi BLOCK. INC e definit ca mai jos.

```
     INC        MACRO     F1,F2
  $ SAVE  GEN
     F1&F2:     DB        0        ;genereată etichete şi
  RESTORE                           DB-uri
                ENDM
```

BLOCK, care acceptă un număr de DB-uri ce urmează a fi generate (NUMB) şi un prefix de etichetă (PREFIX), e definită ca mai jos :

```
        BLOCK   MACRO       NUMB,PREFIX
      $ SAVE  NOGEN
        COUNT   SET         0
                REPT        NUMB
        COUNT   SET         COUNT+1
                INC         PREFIX,%COUNT  ; apelare macră
                                             încorporată
                ENDM
      $ RESTORE
                ENDM
```

Apelarea macro BLOCK 3, LAB generează următorul cod sursă :

```
            BLOCK       3,LAB
    LAB1:   DB          0
    LAB2:   DB          0
    LAB3:   DB          0
```

Controalele de asamblor specificate în aceste două macro (liniile începînd cu $) sînt folosite pentru facilitarea citirii listei de asamblare.

Codul sursă arătat pentru apelarea lui BLOCK 3,LAB e cel ce apare în lista de asamblare cînd sînt folosite controalele. Fără controale lista de asamblare apare ca mai jos:

```
            BLOCK       3,LAB
    COUNT   SET         0
            REPT        3
    COUNT   SET         COUNT+1
            INC         LAB,%COUNT
            ENDM
    COUNT   SET         COUNT$1
            INC         LAB,%COUNT
    LAB1:   DB          0
    COUNT   SET         COUNT+1
            INC         LAB,%COUNT
    LAB2:   DB          0
    COUNT   SET         COUNT+1
            INC         LAB,%COUNT
    LAB3:   DB          0
```

Exemplul 3 :

O macro care se autoconverteşte într-o subrutină.

In anumite cazuri, codarea în - linie (în linie) substituită pentru fiecare apelare macro impune o cerinţă nepermisă de memorie. Următoarele trei exemple arată trei diferite metode pentru conversia unei apelări macro într-o apelare de subrutină. Cînd macro-ul SBMAC e apelat prima dată, are loc o substituire în linie completă care defineşte subrutina SUBR. Fiecare apelare subsecventă al lui SBMAC generează numai o instrucţiune CALL la subrutina SUBR.

In exemplele următoare, reţineţi, că eticheta SUBR trebuie să fie întreagă ca să poată fi apelată din exteriorul primei expansiuni. Acest lucru e posibil numai cînd partea din definiţia macro conţinînd eticheta întreagă,e apelată numai o singură dată în întregul program.

Metoda 1: Definirea macro încorporate

Macro pot fi redefinite în cursul programului. In exemplul următor defimiţia lui,SBMAC conţine propria sa redefinire ca o macro încorporată. Prima dată cînd SBMAC e apelată, e expandată în întregime, şi redefinirea lui SBMAC înlocuieşte definirea originală. A doua oară cînd se apelează SBMAC, numai propria ei redefinire (o instrucţiune CALL) e expandată.

```
SBMAC       MACRO
SBMAC       MACRO
            CALL    SUBR    ;; redefinirea lui SBMAC
            ENDM
            CALL    SUBR
LINK:       JMP     DUN
SUBR:       .
            .
            .
            RET
DUN:
            ENDM
```

Reţineţi că ambele versiuni de SBMAC conţin instrucţi - unea CALL SUBR. Acest lucru e necesar pentru a furniza o adresă de reîntoarcere la sfîrşitul rutinei SUBR.

Instrucţiunea de salt etichetată LINK e necesară pentru a împiedica subrutina SUBR să execute o revenire la eă însăşi.

Reţineţi că adresa de revenire la a doua instrucţiune CALL SUBR trebuie să fie SUBR dacă instrucţiunea de salt a fost omisă. Instrucţiunea JMP DUN transferă controlorul după terminarea subrutinei.

Observaţie : Asamblorul facilitează folosirea liniei sursă constituită numai dintr-o etichetă. O asemenea etichetă e asigurată următoarei linii sursă pentru care s-a generat cod sau dată. Reţineţi că pentru ENDM nu se generează nici cod nici dată, astfel etichetarea DUN e asignată la oricare instrucţiune ce urmează după ENDM. Această construcţie e cerută deoarece SNDM însăşi nu poate furniza o etichetă.

Metoda 2 : Asamblare condiţională

A doua metodă pentru alterarea expansiunii lui SBMAC foloseşte asamblarea condiţionată. In acest exemplu un comutator (FIRST) e poziţionat TRUE chiar înainte de prima apelare pentru SBMAC.SBMAC se defineşte ca mai jos :

```
        TRUE        EQU        OFFH
        FALSE       EQU        0
        FIST        SET        TRUE
        SBMAC       MACRO
                    CALL SIBR
                    IF         FIRST
        FIRST       SET        FALSE
        LINK:       JMP        DUN
        SUBR:       .
                    .
                    .
                    .
                    RET
        DUN:
                    ENDIF
                    ENDM
```

Prima apelare la SBMAC expandează întreaga definiţie, incluzînd apelarea şi definirea lui SUBR:

```
          SBMAC
          CALL       SUBR
          IF         FIRST
LINK:     JMP        DUN
SUBR:     .
          .
          .
          .
          RET
DUN:
          ENDIF
```

Din cauză că FIRST este TRUE, cînd se întîlnește de-a lungul primei expandări a lui SBMAC, toate afirmațiile dintre IF și ENDIF sînt asamblate în program.

In apelări subsecvente, codul asamblat condițional e sărit astfel încît subrutina nu e regenerată. Se produce numai următoarea expandare :

```
SBMAC
CALL       SUBR
IF         FIRST
```

Metoda 3: Asamblare condiționată cu EXITM

Metoda a treia pentru expansiune alterată a lui SBMAC folosește de asemenea asamblarea condiționată, dar utilizează EXITM pentru a suprima expansiunea macro nedorită după prima apelare. EXTIM are efect, cînd FIRST e FALSE, ceea ce are loc după prima apelare la SBMAC.

```
TRUE       EQU        OFFH
FALSE      EQU        O
FIRST      SET        TRUE
SBMAC      MACRO
           CALL       SUBR
           IF         NOT FIRST
           EXITM
           ENDIF
FIRST      SET        FALSE
           JMP        DUN
SUBR       :
           RET
DUN:       ENDM
```

Exemplul 4   Computed GOTO Macro

Această mostră de macro prezintă o implementare a lui GOTO pentru 8080, 8085. GOTO, o caracteristică a multor limbaje de nivel superior, facilitează programului saltul la una dintre diferitele locaţii, ce depind de valoarea variabilei. De exemplu dacă variabila are valoarea zero, programul sare la prima unitate de informaţie din listă. Dacă variabila are valoarea 3, programul sare la a patra adresă din listă. In acest exemplu variabila e plasată în acumulator. Lista de adrese e definită ca o serie de pseudoinstrucţiuni DW, începînd cu adresa simbolică TABLE. Această macro (TJUMP) se automodifică cu o definiţie incorporată. Prin urmare, numai prima apelare la TJUMP generează rutina GOTO. Apelările subsecvente provoacă numai instrucţiunea de salt JMP TJCODE.

```
TJUMP     MACRO               ; stări la adresa A din tabel
TJCODE:   ADD.      A         ; multiplică A cu  2
          MVI       D,0       ;şterge registrul D
          MOV       E,A       ;GET TABLE OFFSET INOT D&E  pune
                               "restul" din tabel în D&E
          DAD       D         ;ADD OFFSET TO TABLE ADDR IN H&L
                               adună offsetul la adresa din H şi
                               H a tabelului
          MOV       E,M       ;GET-găseşte primul byte de adresă
          INX       H
          MOV       D,M       ;găseşte al doilea byte de adresă
          XCHG
          PCHL                ;sari la adresa
RJUMP     MACRO               ;REDEFINESTE TJUMP pentru salvarea
                               codului
          JMP       TJCODE    ; următoarea apelare sare la codul
                                anterior (JUMPS TO ABOVE CODE)
          ENDM
          ENDM
```

Reţineţi că definirea lui TJUMP nu trebuie să justifice încărcarea adresei tabelului de adrese în registrele H şi L; utilizatorul trebuie să încarce aceste adrese chiar înainte de apelarea lui TJZMP.

In cele ce urmează se prezintă codarea tabelului de adrese (TABLE) şi o secvenţă tipică de apelare pentru macro-ul TJUMP :

```
          MVI        A,2
          LXI        H,TABEL
          TJUMP
          .
          .
          .
          .
TABLE:    DW         LOC0
          DW         LOC1
          DW         LOC2
```

Secvenţa de apelare prezentată mai sus provoacă un salt la LOC2.

Exemplul 5 : Folosirea lui IRP pentru definirea tabelului de salt

Macro TJUMP devine chiar mai folositoare cînd o a doua macro (GOTO) e folosită pentru a defini tabelul de salt, se încarcă adresa tabelului în registrele H şi L şi apoi apelează TJUMP. Macro GOTO e definită ca mai jos :

```
GOTO      MACRO      INDEX, LIST
          LOCAL      JTABLE
          LDA        INDEX          ; încarcă acumulatorul
                                      cu INDEX
          LIX        H,JTABLE       ; încarcă H&L cu adresa
                                      tabelului
          TJUMP                     ; apelează macro TJUMP
JTABLE:   IRP        FORMAL, LIST
          DW         FORMAL
          ENDM
          ENDM
```

O apelare tipică pentru GOTO poate fi de forma :

```
GOTO   CASE,  COUNT,TIMER, DATE,PTDRVR
```

Această apelare la GOTO construieşte un tabel de pseudoinstrucţiuni DW pentru etichetele COUNT, TIMER, DATE şi PTDRVR. Apoi încarcă adresa de bază a tabelului în registrele H&L şi apelează RJUMP. Dacă valoarea variabilei CASE e 2 cînd se apelează GOTO, macro-urile GOTO şi TJUMP împreună provoacă un salt la adresa rutinei DATE.

Reţineţo că oricîte adrese pot fi specificare în liste pentru rutina GOTO atît timp cît încap toate pe o singură linie sursă. De asemenea GOTO poate fi apelat ori de cîte ori, dar numai o singură copie a codării pentru TJUMP e generată pînă cînd TJUMP se autoredefineşte pentru a genera numai o instrucţiune JMP TJCODE.

6. TEHNICI DE PROGRAMARE

(Branch Tables Pseudo-Subroutine)

Pseudo- Subrutine cu tabele de ramificaţie

Presupunem că un program se compune din mai multe rutine separate, execuţia fiecăreia din acestea depinzînd de anumite condiţii iniţiale (ca de exemplu un număr trecut în registru). O cale de codare este aceea de a controla fiecare din aceste condiţii în mod secvenţial şi de a ramifica spre rutine ca mai jos :

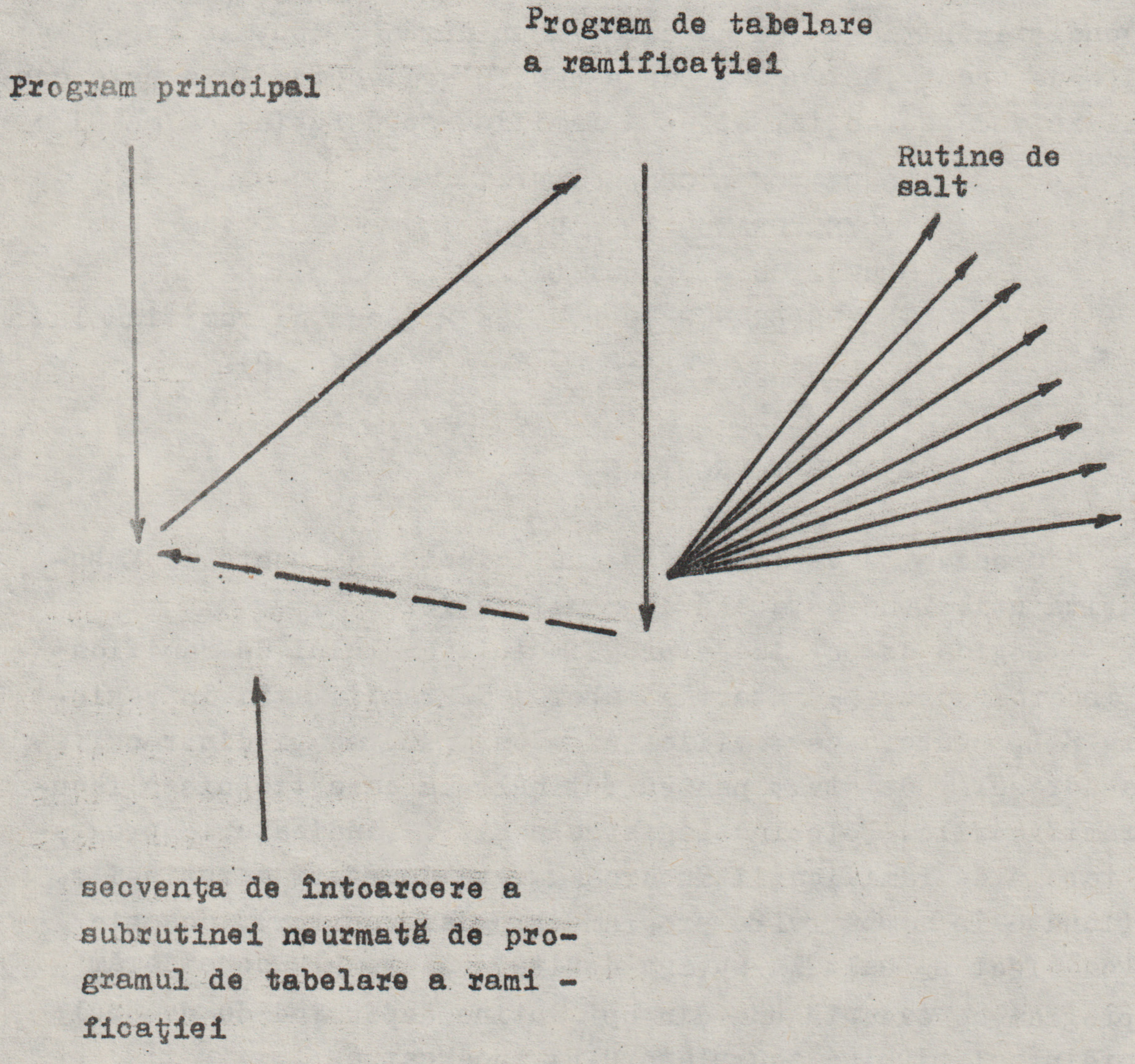
```
CONDITION = CONDITION1 ?
IF YES BRANCH TO ROUTINE 1
CONDITION = CONDUTION 2 ?
IF YES BRANCH TO ROUTINE 2 (dacă da ramifică la
.                           rutina 2)
.
.
BRANCH TO ROUTINE N
```

O secvenţă ca mai sus e ineficientă, şi poate fi îmbunătăţită utilizînd o tabelă de ramificaţie.

Logica de pornire a programului tabelului de ramificaţii încarcă adresa de start a tabelei de ramificaţii în registrele H&L, Tabela de ramificaţii constă ea însăşi dintr-o listă a adreselor de start pentru rutinele la care trebuiesc făcute ramificările. Folosind registrele H&L ca indicatori, programul tabelă de ramificaţii încarcă adresa de start a rutinei selecţionate în numărătorul de program (progran counter), aceasta are ca efect un salt la rutina dorită. De exemplu considerăm un program ce execută una din opt rutine depinzînd de numărul de ordine al bitului acumulatorului încărcat cu 1 :

Salt	la	rutina	1	dacă	acumulatorul	are	starea	00000001
"	"	"	2	"	"	"	"	00000010
"	"	"	3	"	"	"	"	00000100
"	"	"	4	"	"	"	"	00001000
"	"	"	5	"	"	"	"	00010000
"	"	"	6	"	"	"	"	00100000
"	"	"	7	"	"	"	"	01000000
"	"	"	8	"	"	"	"	10000000

Urmează programul ce furnizează o asemenea logică. Programul se numeşte 'pseudo subrutină' pentru că e tratat de programator ca o subrutină (adică, apare numai o dată în memorie), dar e angajat mai curînd cu o instrucţiune JUMP decît ca una CALL.



```
Eticheta    Code    Operand
START:      LXI     H,BTBL      ; registrul H&L vor
                                ;controla tabela de ramifi-
                                 caţii
GTBIT:      RAR
            JC      GETAD
            INX     H           ;(H,L)=(H,L)+2 pentru
            INX     H           ;a controla următoarea adresă
                                ;în tabela de ramificaţii
            JMP     GRBIT
GETAD:      MOV     E,M         ;găsirea bitului
            INX     H           ;încărcarea adresei de salt
                                ;în registrul D&E
            MOV     D,M
            XCHG                ;schimbă D&E
                                ; cu H&L
            PCHL                ;salt la adresa rutinei
            .
            .
            .
BTBL:       DW      ROUT1       ;Tabela de ramificaţii.Fiecare
            DW      ROUT2       ;intrare e o adresă
            DW      ROUT3       ;de doi byte
            DW      ROUT4       ;(HEID) pune cel mai puţin
                                ;semnificativ
            DW      ROUT5       ;byte-primul
            DW      ROUT6
            DW      ROUT7
            DW      ROUT8
```

La START rutina de control foloseşte registrele H&L ca indicatori în tabela de ramificaţii (BTBL) corespunzînd cu bitul setat din acumulator. Rutina la GETAD transferă adresa ţinută la intrarea corespunzătiare a tabelei de ramificaţii în registrele H&L prin registrele D&E, şi apoi utilizează o instrucţiune PCHL care transferă controlul la rutina selectată.

<u>TRANSFERUL DE DATE LA SUBRUTINE</u>

O subrutină cere în mod tipic date pentru realizarea propriile-i operaţii. In cazul cel mai simplu, aceste date pot fi transferate în unul sau mai multe registre.

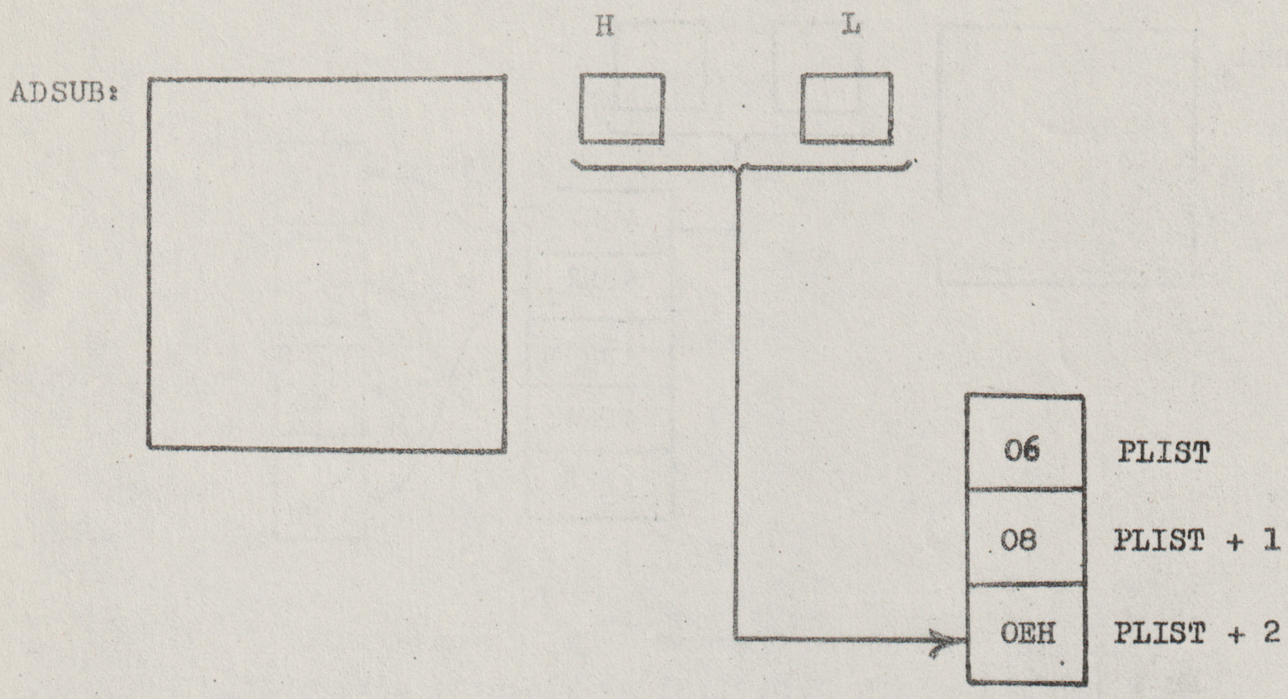
Uneori e mai convenabil şi mai economic să lăsăm subrutina să-şi încarce propriile registre. O cale de a face acest lucru e, de a plasa o listă cu datele cerute (numită listă de parametrii) în zona de date a memoriei, şi de a trece adresa acestei liste subrutinei în registrele H&L.

De exemplu subrutina ADSUB aşteapză adresa unei liste de parametrii de trei byte în registrele H&L. Ea adună primul şi al doilea byte al listei, şi stochează rezultatul în al treilea byte al listei :

```
Etichetă   Code   Operand    Comentariu
           LXI    H,PLIST    ;=încarcă H&L cu
                             ;adresele listei de parametrii
           CALL   ADSUB      ;cheamă subrutina
RET1:
PLIST:     DB     6          ;primul număr de adunat
           DB     8          ;al doilea număr de adunat
           DS     1          ;rezultatul va fi stocat aici
           LXI    H,LIST2    ;încarcă registrele H&L
           CALL   ADSUB      ;pentru o altă apelare a ADUSB
RET2:
LIST2:     DB     10
           DB     35
           DS     1
           %
           •
           •
ADSUB:     MOV    A,M        ;ia primul parametru
           INX    H          ;incrementează adresa memoriei
           MOV    B,M        ;ia al doilea parametru
           ADD    B          ;adună primul la al doilea
           INX    H          ;incrementează adresa memoriei
           MOV    M,A        ;stochează rezultatul la al
                              treilea byte
           RET               ;revenire necondiţionată
```

Prima dată cînd ADSUB e apelat, el încarcă registrele A&B cu PLIST şi PLIST+1. le adună, şi stochează rezultatul în PLIST+2. Revenirea se face la instrucţiunea RET1.

Prima apelare şa ADSUB :



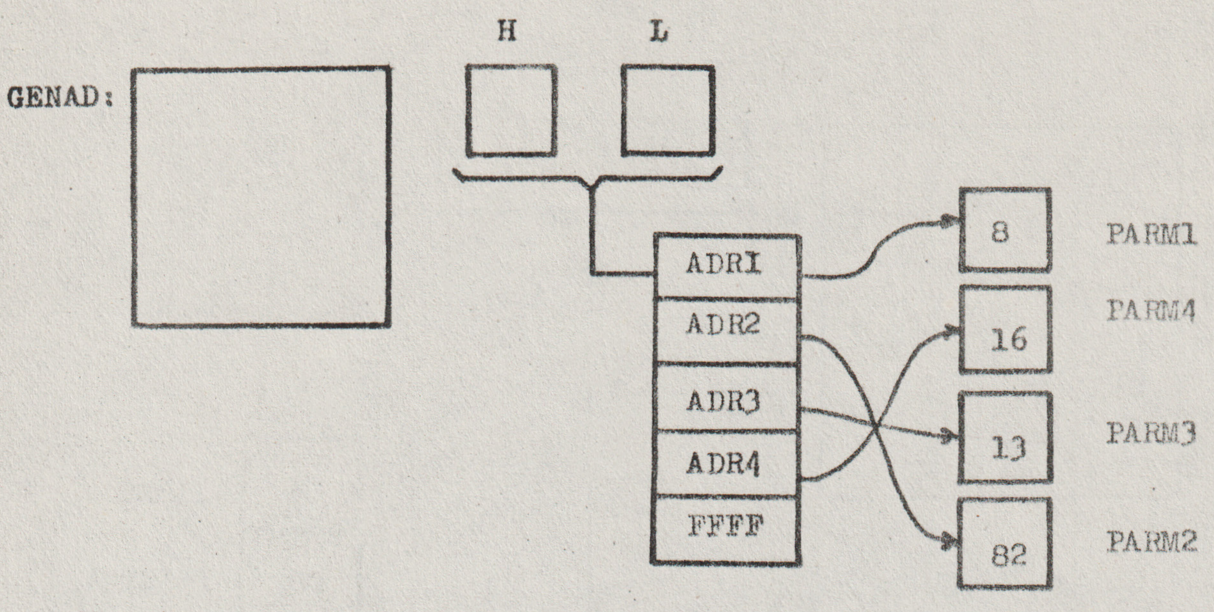
La a doua apelare a ADSUB, registrele H&L indică lista de parametrii LIST 2, Registrele A&B sînt încărcate cu 10 şi respectiv 35, şi suma e stocată la LIST2+2. Revenirea se face la instrucţiunea RET2.

Reţinei că listele de parametrii PLIST şi LIST2 pot apare oriunde în memorie fără a altera rezultatul produs de ADSUB.

Acest mod totuşi îşi are limitele lui. Odată codat, ADSUB trebuie să primească două şi numai două numere de adunat, şi ele trebuie să fie continue în memorie. Presupunem că dorim o subrutină (GENAD) care va aduna un număr arbitrar de byte-ţi localizatăoriunde în memorie, şi să lase suma în acumulatro.

Acest lucru poate fi făcut trecînd subrutinei o listă de parametrii care e o listă a adreselor parametrilor, mai degrabă decît a parametrilor înşişi, şi însemnînd că sfîrşitul listei de parametrii e un număr al cărui prim byte e FFH (presupunîns că deasupra adresei FFOOH nu vor fi stocaţi patametri.

Apelarea la GENAD :




```
Etichetă      Cod     Operand      Comentariu
              LXI     H,PLIST      ;încarcă adresa în
              CALL    GENAD        ;lista de adrese a parametrilor
              .
              .
              HALT
PLIST:        DW      PARAM1       ;lista adreselor parametrilor
              DW      PARAM2
              DW      PARAM3
              DW      PARAM4
              DW      OFFFFH       ;terminator
              .
              .
PARAM1:       DB      6
              DB      16
              .
              .
PARAM3:       DB      13
              .
              .
              .
PARAM2:       DB      82
              ..
              .
GENAD:        XRA     A            ;şterge acumulatorul
LOOP:         MOV     C,A          ;salvează suma curentă în C
              MOV     E,M          ;ia byt-ul de ordine inferioară
                                   ;al primului parametru
              INX     H
```

```
        MOV     A,M       ;ia byte-ul de ordin superior al pri-
                          mului parametru
        CPI     OFFH      ;compară cu FFH
        JZ      BACK      ;dacă e egal, rutina e completă
        MOV     D,A       ;D&E adresează acum parametrul
        LSAX    D         ;încarcă acumulatorul cu parametru
        ADD     C         ;adună suma anterioară
        INX     H         ;incrementează H&L pentru a indica
                          ;adresa parametrului următor
        JMP     LOOP      ;ia următorul parametru
BACK:   MOV     A,C       ;rutină termină-restichează totalul
        RET               ;revenire la rutina apelată
        END
```

Reţineţi că GENAD poate aduna orice combinaţie de parametri fără a schimba parametrii respectivi

Secvenţa :

```
        LXI     H,PLIST
        CALL    GENAD
        .
        .
        .
PLIST:  DW      PARM4
        DW      PARM1
        DW      OFFFFH
```

va cauza adunarea lui PARM1 şi PARM4, putînd fi oriunde localizaţi în memorie (excluzînd adresele deasupra de FFOOH)

Sînt posibile multe variante de transfer de parametrii. De exemplu e necesar să stocăm parametrii la o adresă, un program de apelare poate trece numărul total de parametrii ca primul parametru; subrutina încarcă apoi primul parametru,într-un registru şi îl foloseşte drept contor pentru a determina cînd toţi parametrii au fost aceptaţi.

Multiplicare şi divizare software

Inmulţire şi împărţire - software

Inmulţirea a 2 byte-ţi de date poate fi realizată prin 2 metode: adunarea repetată sau folosindu-se o operaţie de deplasare de registru.

Adunarea repetată este simplă dar lentă. De exemplu 2AH*75H se poate face adunînd 75H de 2AH ori.

Operaţiile de deplasare realizează o multiplicare mai rapidă. Deplasînd un byte la stînga cu un bit e echivalent cu a-l multiplica cu 2, iar deplasîndu-l la dreapta cu un bit e echivalent cu a-l împărţi cu 2. Pentru înmulţirea a 2 byte-ţi trebuie să aibă loc următorul proces :

A. Se testează cel mai puţin semnificativ bit al înmulţitorului. Dacă e 0, se trece la pasul B. Dacă e 1, se adună deînmulţitul cu cel mai semnificativ byte al rezultatului.

B. Se deplasează tot rezultatul cu un bit la dreapta.

C. Se repetă A şi B pînă cînd se testează toţi biţii înmulţitului.

De exemplu, să considerăm : 2AH*3CH = 9D8H

Pasul 1 : Testul bitului 0 al îmulţitorului ; e 0, deci se deplasează rezultatul de 16 bit cu un bit spre dreapta.

Pasul 2 : Testul bitului 1 al înmulţitorului, e 0, deci se deplasează rezultatul la dreapta cu un bit.

Pasul 3 : Testul bitului 2 al înmulţitorului ; e 1, deci se adună 2AH la byte-ul cel mai semnificativ al rezultatului şi se deplasează rezultatul la dreapta cu un bit.

Pasul 4 : Bitul 3 al multiplicatorului de test; e 1 deci adună 2AH la byte-ul de ordin superior a rezultatului şi deplasează rezultatul de 16 bit la dreapta cu un bit.

Pasul 5 : Bitul 4 al multiplicatorului de test; e 1; deci adună 2AH la byte-ul de ordin superior a rezultatului şi deplasează rezultatul de 16 bit la dreapta cu un bit.

Pasul 6 : Bitul 5 al multiplicatorului de test; e 1; deci adună 2AH la byte-ul de ordin siperior a rezultatului şi deplasează rezultatul de 16 bit la dreapta cu un bit.

Pasul 7 : Bitul 6 al multiplicatorului de test, e 0; deci deplasează rezultatul de 16 bit la dreapta cu un bit.

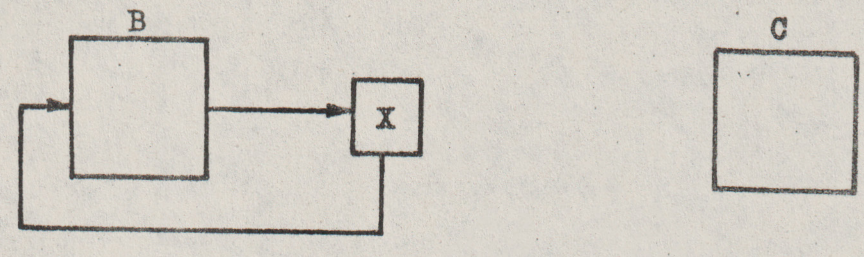
Pasul 8 : Bitul 7 al multiplicatorului de test; e 0; deci deplaseată rezultatul de 16 bit la dreapta cu un bit. Rezultatul produs e 09D8.

Postul 1	a		
	b	00000000	00000000
Postul 2	a		
	b	00000000	00000000
Postul 3	a	00101010	00000000
	b	00010101	00000000
Postul 4	a	00111111	00000000
	b	00011111	10000000
Postul 5	a	01001001	10000000
	b	00100100	11000000
Postul 6	a	01001110	11000000
	b	00100111	01100000
Postul 7	a		
	b	00010011	10110000
Postul 8	a		
	b	00001001	11011000(9D8)

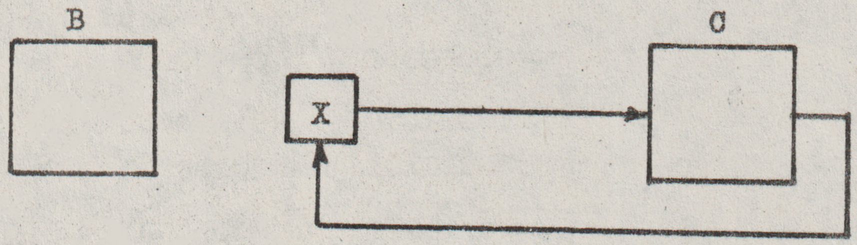
Deoarece rutina de multiplicare descrisă mai sus foloseşte un număr important de tehnici de programare, un exemplu de program e dat cu comentarii.

Programul foloseşte registrul B pentru a înregistra cel mai semnificativ byte al rezultatului şi registrul C pentru cel mai puţin semnificativ byte. Deplasarea la dreapta a celor 16 biţi ai rezultatului se realizează în acumulator de două instrucţiuni de rotire prin bitul CY la dreapta (rotate-right-through-carry instrucţioun).

Resetează bitul CY şi apoi rotează B :



Apoi rotează C pentru a completa deplasarea :



Registrul D ţine deînmulţitul şi registrul C iniţial ţine înmulţitorul

```
MULT:     MVI   B,0     ; iniţiază cel mai semnificativ byte
                          al rezultatului
          MVI   E,9     ; numărător de bit
MULTO:    MOV   A,C     ; deplasează cel mai puţin semnificativ
          RAR           ; bit al multiplicatorului spre CY şi
          MOV             deplasează
          MOV   C,A     ; byte-ul de ordin inferior al rezulza-
                          tului
          DCR   E
          JZ    DONE    ; ieşire dacă e complet
          MOV   A,B
          JNC   MULTI
          ADD   D       ; adună deînmulţitul la byte-ul de ordin
                        ; superior al rezultatului dacă bitul a
                        ; fost un 1
MULTI:    RAR           ; CARRY=0 aici deplasează byte-ul de or-
                        ; din superior al rezultatului
          MOV   B,A
          JMP   MULTO
DONE:
```

O procedură analoagă e folosită la împărţirea unui număr de 16 biţi fără semn, cu un altul de 16 biţi tot fără semn. Aici, procesul implică mai degrabă scădere decît adunare şi instrucţiunea de rotare la stînga în loc de rotare la dreapta.

Următorul program de reintrare foloseşte registrele B&C pentru a ţine deîmpărţitul şi cîtul şi registrele D&C pentru a ţine împărţitorul şi restul.

Registrele H&L sînt folosite pentru a stoca temporar data.

```
DIV:  MOV   A,D    ; neagă divizorul
      CMA
      MOV   D,A
      MOV   A,E
      CMA
      MOV   E,A
      INX   D      ;pentru complementul lui doi
      LXI   H,0    ;valoarea iniţială pentru rest
      MVI   A,17   ;iniţializarea numărătorului de buclă
DVO:  PUSH  H      ;salvează restul
      DAD   D      ;scade divizorul (adună negativ)
      JNC   DVI    ;sub debit, restochează HL
      XTHL
DV1:  POP   H
      PUSH  PSW    ;salvează numărătorul de buclă (A)
      MOV   A,C    ;patru deplasări de registru la stînga
      RAL          ;cu transport
      MOV   C,A    ; CY ->C ->B ->L ->H
      MOV   A,B
      RAL
      MOV   B,A
      MOV   A,L
      RAL
      MOV   L,A
      MOV   A,H
      RAL
      MOV   H,A
      POP   PSW    ;restochează numărătorul de buclă(A)
      DCR   A      ;decrementează - 1
      JNZ   DVO    ; ţine bucla
```

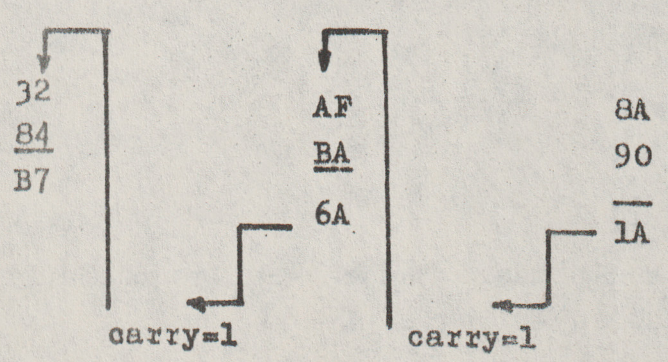
```
; şterge postdiviziunea
; deplasează restul la dreapta şi revine în DE
;
        ORA     A
        MOV     A,H
        RAR
        MOV     D,A
        MOV     A,L
        RAR
        MOV     E,A
        RET
        END
```

Adunarea şi scăderea multibyte

Indicatorul de transport şi instrucţiunile ADC (adunare cu transport) por fi utilizate pentru a aduna datele fără semn de lungimi arbitrare. Considerăm următoarea adunare a două numere hexa fără semn, de trei byte :

```
  32AF8A
+ 84BA90
  ------
  B76A1A
```

Realizarea acestei adunări: adună byte-ul de ordin inferior cu instrucţiunea ADD. Acesta setează indicatorul de transport pentru folosire în instrucţiunile subsecvente, dar nu include indicatorul de transport în adunare. Apoi foloseşte ADC pentru adunarea tuturor byte-urilor de ordin superior.

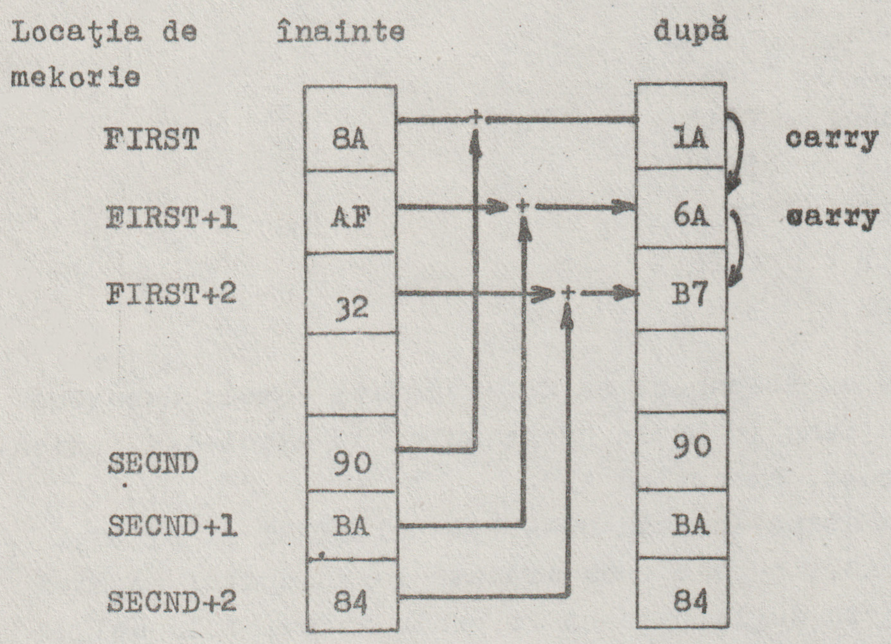


Următoarea rutină va realiza această adunare multibyte făcînd următoarele presupunei:

In registrul E se stochează lungimea fiecărui număr de adunat (în acest caz 3).

Numerele de adunat sînt stocate de la byte-ul de ordin inferior la byte-ul de ordin superior în locaţiile de memorie FIRST şi SECND.

Retultatul va fi stocat de la byte-ul de ordin inferior la cel de ordin superior la adresa FIRST din memorie,înlocuind conţinutul original al acestei locaţii.



Următoarea rutină foloseşte o instrucţiune ADC pentru byte-ul de ordin inferior al operanzilor. Aceasta poate cauza ca rezultatul să fie mărit cu unu dacă indicatorul de transport a fost setat de o instrucţiune anterioară.

Această rutină evită problema ştergînd indicatorul de transport cu instrucţiunea XRA chiar înainte de LOOP.

```
Eticheta    Cos    Operand     Comentariu

MADD:       LXI    B,FIRST     ;B&C adresează pe FIRST
            LXI    H,SCND      ;H&L adresează pe SCND
            XRA    A           ;şterge indicatorul de transport
LOOP:       LDAX   B           ;încarcă byte-ul lui FIRST
            ADC    M           ;adună byte-ul la SCND cu trans-
                                port
            STAX   B           ;stochează rezultatul la FIRST
            DCR    E           ;terminat dacă IF E=0
            INX    B           ;indică byte-ul următor al lui
                                FIRST
            INX    H           ;indică următorul byte al SCND
            JMP    LOOP        ;adună următorii doi byte
DONE:       .
            .
FIRST:      DB     90H
            DB     0BAH
            DB     84H
SCND:       DB     8AH
            DB     0AFH
            DB     32H
```

Adunarea cu transport va fi executată corect,deoarece nici o instrucţiune în bucla programului nu afectează indicatorul de transport, exceptînd ADC.

Cînd s-a atins locaţia DONE,byte-ul FIRST şi FIRST+2 vor conţine 1A6AB7, care e suma arătată la începutul acestei secţiuni, aranjată de la byte-ul de ordin inferior la cel de ordin superior.

Pentru a creea o rutină de scădere multibyte, e necesară numai multiplicarea rutinei de adunare multibyte a acestei secţiuni, schimbînd instrucţiunea ADC cu una SBB. Programul va scade apoi următorul început la SCND din cel început la FIRST, punînd rezultatul la FIRST.

Adunarea zecimală

Orice cantitate de date de 4 biţi poate fi tratată ca un număr zecimal atît timp cît aceasta reprezintă unul din cifrele zecimale 0-9, şi să nu conţină nici un bit caracteristic reprezentării cifrelor zecimale în nexa A la F. Pentru a re -

zerva această interpretare zecimală cînd realizăm adunarea, valoarea 6 trebuie adunată cantităţii de 4 biţi ori de cîte ori adunarea produce un rezultat între 10 şi 15. Aceasta pentru că fiecare cantitate de dată de 4 biţi poate conţine cu 6 combinaţii mai mult decît ar fi necesar pentru cifrele zecimale.

Adunarea zecimală e realizată lăsînd ca fiecare byte ((8 biţi) să reprezinte două cifre zecimale de 4 biţi. Byte-urile sînt însumate în acumulator în formă standard şi instrucţiunea DAA e apoi folosită la convertirea rezultatului binar de 8 biţi într-o reprezentare de două cifre zecimale. Pentru linii multibyte, trebuie să realizaţi ajustarea zecimală înainte de adunarea următorului byte de ordin superior. Aceasta pentru că aveţi nevoie de indicatorul de transport de la instrucţiunea DAA a fi adunat byte-ul de ordin superior.

Pentru realizarea adunării zecimale :

```
  2985
+ 4935
------
  7921
```

se întîmplă următoarele :

1. Se şterge Carry şi se adună cele două cifre de ordin inferior a fiecărui număr (reţinei că fiecare două cifre zecimale sînt reprezentate de un byte)

```
             85  =  10000101B
             36  =  00110110B
          carry  =          0
                 ------------
                 [0] 10111011B

Carry=0                      Auxiliary Carrx=0
```

Aculululătorul conţine acum OBBH.

2. Realizarea operaţiei DAA. Dacă cei mai din dreapta patru biţi sînt mai mari decît 9, se adună un 6 la acumulator

```
Acumulator  =  10111011B
         6  =      0110B
               ---------
               11000001B
```

Dacă cei mai din stînga 4 biţi sînt mai mari decît 9 e adunat un 6 la aceşti biţi, setîndu-se astfel indicatorul de transport.

```
Acumulator  =  11000001B
         6  =  0110    B
              ----------
              [1] 00100001B
               ↑
Carry Flag=1
```

Acumulatorul conţine acum 21H se stochează aceste două cifre.

3. Adună grupul următor de două cifre :

```
        29  =  00101001B
        49  =  01001001B
  carry     =         1
              ----------
              [0] 01110011B
              ↗      ↖
      Carry=0         Auxiliary Carry=1
```

acumulatorul conţine acum 73H.

4. Realizează o operaţie DAA. Dacă indicatorul auxiliar de transport e setat se adună 6 la acumulator.

```
Acumulator =   01110011B
         6 =       0110B
              ----------
              [0] 01111001B
               ↑
   Carry Flag=0
```

Dacă cei mai din stînga patru biţi sînt mai mici de 10 şi indicatorul de transport e resetat, nu mai intervine alte acţiuni.

Deci rezultatul zecimal corect va fi generat în doi byte (7921) .

O rutină care adună numele zecimal, e analoagă cu rutina de adunare multibyte MADD a ultimei secţiuni, şi poate fi produsă înserînd instrucţia DAA după instrucţiunea ADC M a exemplului dat.

Fiecare iteraţie a buclei de program va aduna cifre zecimale (un byte).

Scădere zecimală

Scăderea zecimală e considerabil mai complicată decît adunarea zecimală. In general, procesul constă în generarea complementului faţă de 10 a cifrei care se scade, şi apoi adunînd rezultatul la cifra de scăzut. De exemplu: scădem 34 din 56, formăm complementul faţă de 10 a lui 34 (99-34=65+1=66). Deci 56+66=122. Separînd transportorul din cifra de ordin superior vom găsi rezultatul corect 22 .

Problema împrumutului apare în scăderea zecimală multibyte. Cînd nu e nevoie de împrumut, veţi folosi complementul faţă de 10 a cifrei care se scade, pentru următoarea operaţie. Dacă va apare un împrumut, vei folosi complementul faţă de nouă.

Reţineţi că sensul indicatorului de transport e invers pentru că aţi lucrat cu date complementare. Deci, un bit 1 în indicatorul de transport înseamnă lipsă împrumut, iar un zero în bitul de transport indică împrumut.

Acest indicator de transport inversat poate fi folosit în operaţia de adunare pentru a forma complenehtul faţă de 9 sau 10 al cifrei care se scade.

Procedura detaliată a scăderii numerelor zecimale multidigit :

1. Setarea indicatorului de transport la 1 pentru a indica lipsă de împrumut este următoarea :

2. Incarcă acumulatorul cu 99H, ce reprezintă numărul zecimal 99.

3. Adună zero la acumulator cu transport, producînd sau 99H sau 9AH şi resetînd indicatorul de transport.

4. Scade cifrele descăzutului din acumulator producînd complementul faţă de 9 sau 10.

5. Adună cifra de scăzut la acumulator.

6. Foloseşte DAA pentru a pune sigur rezultatul în acumulator în format zecimal, şi să indice un împrumut în indicator de transport dacă se întîlneşte.

7. Dacă sînt mai multe cifre de scăzut se merge la pasul 2. Astfel STOP.

Exemple :

Realizaţi scăderea zecimală :

```
   4358D
 - 1362D
 -------
   2996D
```

1. Setează transportul = 1
2. Incarcă acumulatorul cu 99H
3. Adună zero la acumulator cu transport rezultînd 9AH

```
Acumulator  = 10011001B
            = 00000000B
Carry       =         1
              ---------
              10011010B = 9AH
```

4. Scade cifra scăzător 62 din acumulator

```
Acumulator  = 10011010B
       ‾‾     
       62   = 10011110B
             ----------
            1]00111000B
```

5. Adună descăzutul 58 la acumulator

```
Acumulator  = 00111000B
        58  = 01011000B
             ----------
            0]10010000B = 90H

Carry=0                  Auxiliar carry=1
```

6. DAA converteşte acumulatorul la 96 (pînă cînd auxiliary carry=1) şi lasă indic atorul de transport = 0, indicînd că un împrumut a avut loc.

7. Incarcă acumulatorul cu 99H.

8. Adună zero cu transport la acumulator lăsînd acumulatorul= 99H

9. Scade scăzătorul 13 din acumulator

```
Acumulator  = 10011001B
       ‾‾
       13   = 11101101B
             ----------
            1]10000110B
```

10. Adună cifrele de scăzur 43 la accumulator

```
          Acumulator = 10000110B
                  43 = 01000011B
                     -----------
                     0 11001001B = C9H

    Carry=0                Auxiliar carry = 0
```

11. DAA converteşte accumulatorul la 29 şi pune bitul CY=1, indicînd că nu a avut loc împrumut.

Prin urmare rezultatul scăderii este 2996.

Subrutina următoare va scădea un număr zecimal de 16 digit, din altul, folosind următoarele presupuneri :

Descăzurul e stocat începînd cu cifrele cele mai puţin semnificative (2) a locaţia MINU.

Scăzătorul e stocat începînd cu cifrele cele mai puţin semnificative (2) la locaţia SBTRA.

Rezultatul va fi stocat, mai întîi cu cifrele cele mai puţin semnificative (2) înlocuind descăzutul.

```
Eticheta   Cod    Operand      Comentariu
DSUB:      LXI    D,MINU
           LXI    H,SBTRA

           MVI    C,8



           STC

LOOP:      MVI    A,99H

           ACI    0
           SUB    M

           XCHG

           ADD    M
           DAA
```

Eticheta	Cod	Operand	Comentariu
	MOV	M,A	
	XCHG		
	DCR	C	
	JZ	DONE	
	INX	D	
	INX	H	
	JMP	LOOP	
DONE:	NOP		

7. INTRERUPERI

Generalităţi

Adesea intervin evenimente externe UP-ului, care cer acţionarea imediată a UCP.

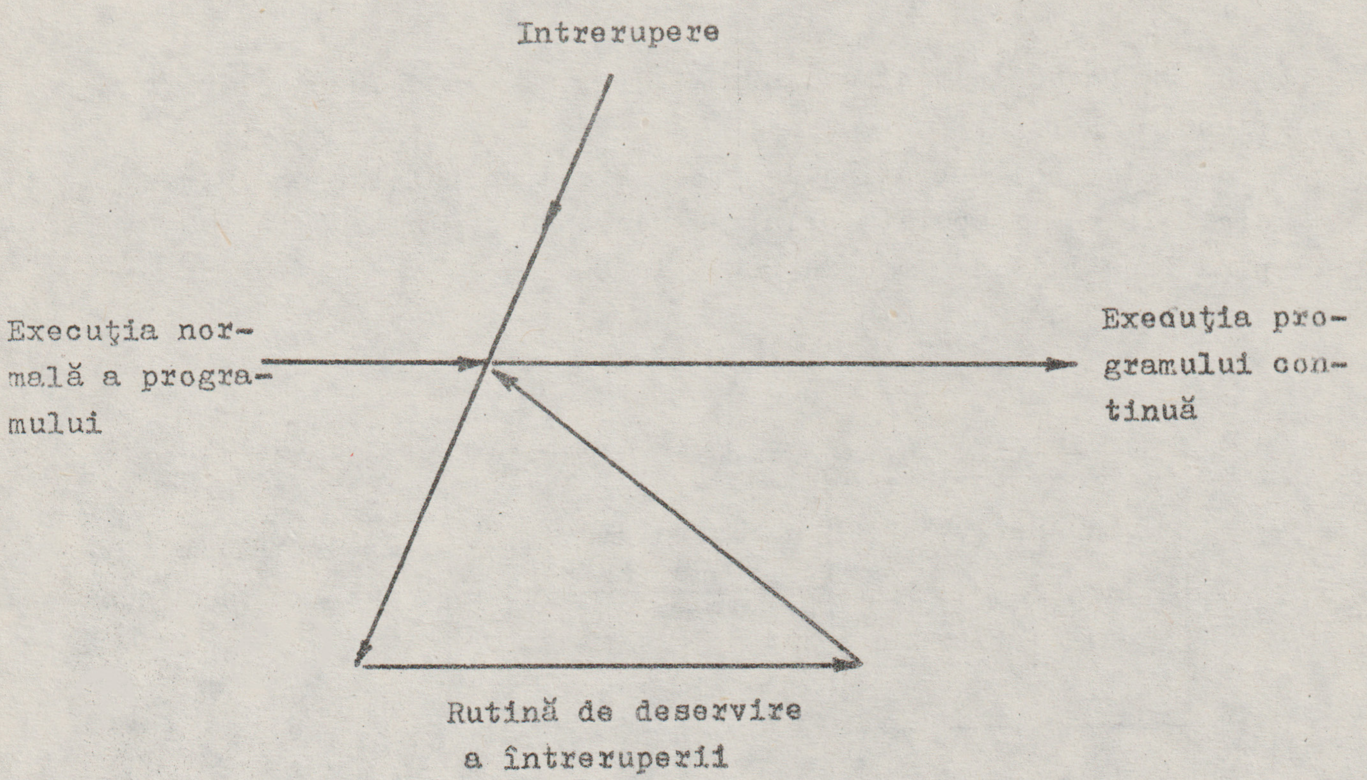
De exemplu presupunem că un echipament trimite un şir de 80 de caractere la UCP, unul cîte unul la intervale de timp constante. Există două căi de deservire a acestei situaţii :

A. Se poate scrie un program care să accepte primul caracter, să aştepte pînă caracterul următor e gata (de exemplu, execută un time out prin incrementarea unui numărător suficient de mare), acceptă apoi caracterul următor .procedează în acest mod pînă ce toate cele 80 de caractere au fost primite.

Această metodă se referă la programarea Input/Output.

B. Circuitul la control al dispozitivului periferic poate intrerupe UCP-ul cînd un caracter e gata să intre, forţînd o ramificaţie de la programul de executat la o rutină specială de întrerupere.

Secvenţa de întrerupere e ilustrată mai jos:



In capitolul 3 se descriu instrucţiunile EI şi DI care pot seta ori reseta bitul numit INTE din 8080. Oricînd INTE=0, întregul sistem de întreruperi e inhibat, şi nu se acceptă întreruperi.

Cînd 8080 recunoaşte o cerere exterioară de întrerupere, lucrările se întîmplă ca mai jos :

1.Se termină instrucţiunea în curs de execuţie

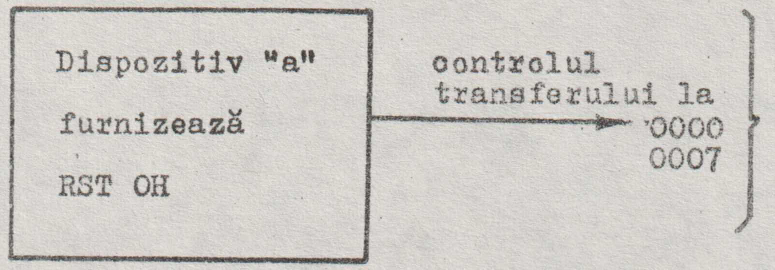
2.INTE=0

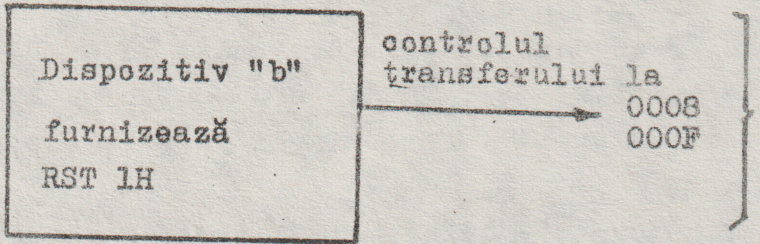
3.Echipamentul care provoacă întreruperea, furnizează, prin hardware. o instrucţie pe care UCP o execută. Această instrucţiune nu apare nicăieri în memorie şi programatorul nu are control asupra ei, deoarece ea este o funcţie a circuitului de control al echipamentului care provoacă întreruperea. Numărătorul de program nu e incrementat înaintea acestei instrucţiuni.

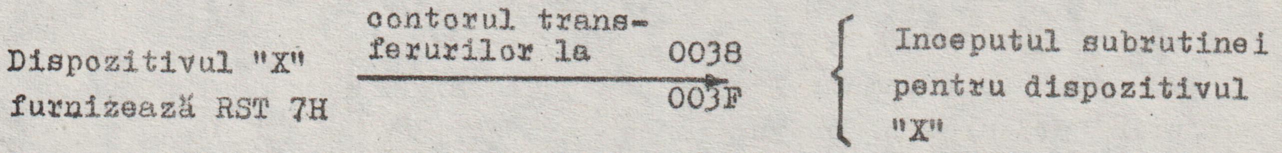
Instrucţiunea furnizată de echipament întreruptor în mod normal e RST care e o subrutină eficientă care cheamă un byte din 8 posibili localizaţi în primele 64 cuvinte de memorie- De exemplu echipamentul exterior poate furniza instrucţiunea: RST OH pentru fiecare întrerupere de intrare. Atunci subrutina care procesează datele transmise de diapozitivul extern la UCP va fi chemată în execuţie printr-o secvenţă de instrucţiuni de 8 byte-ţi la locaţiile de memorie 0000H-0007H.

Un echipament de intrare digital poate furnita instrucţiunea : RST 1H.

Atunci subrutina de procesare a semnalelor de intrare digitale va fi apèlată printr-o secvenţă de instrucţiuni ocupînd locaţiile de memorie 0008H la 000FH.



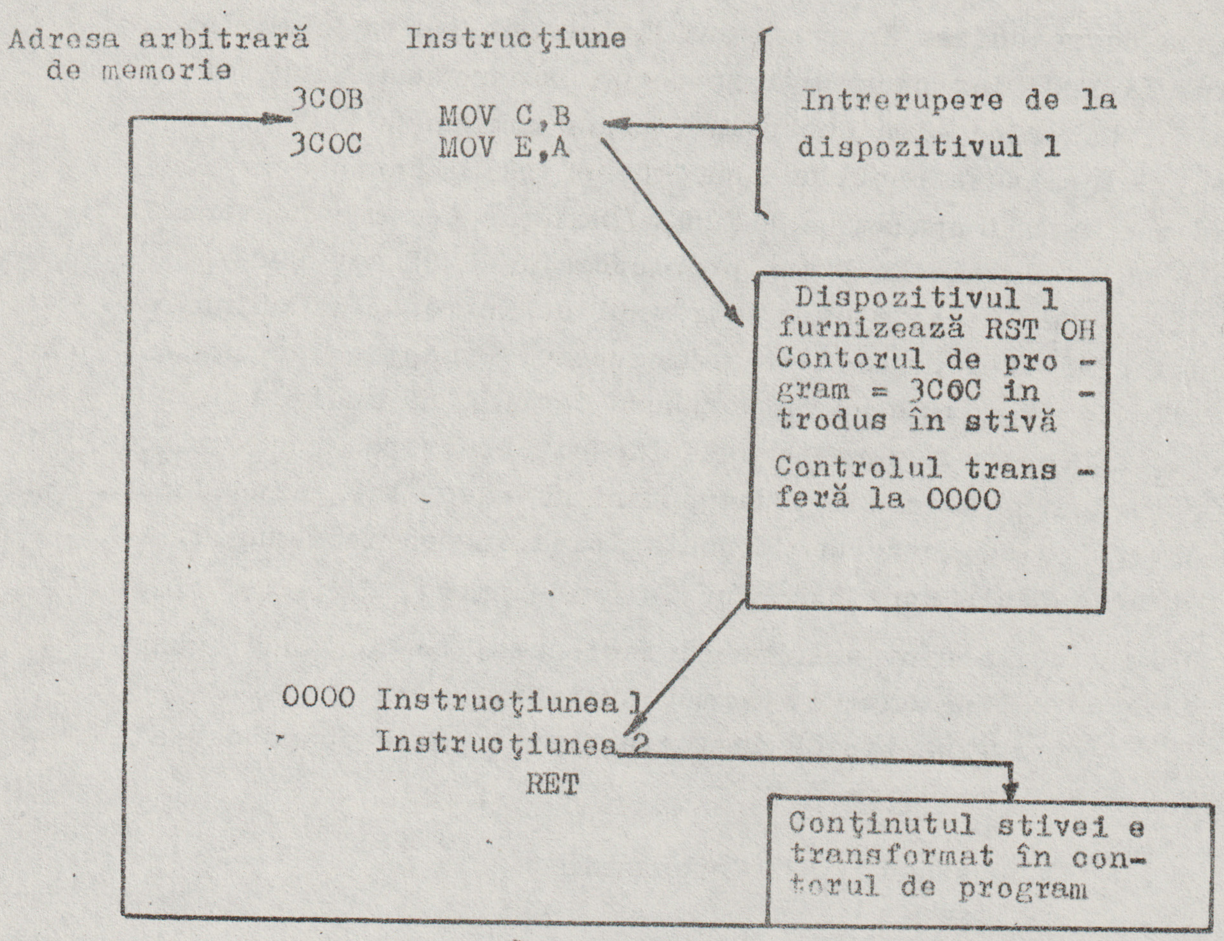




Reţineţi că fiecare dintre aceste subrutine de 8 byte se pot transforma în subrutine mai lungi pentru a procesa întreruperile, dacă e necesar.

Oricare echipament poate funiza înstrucţiuhea RST (şi desigur poate furniza orice instrucţiune 1 byte 8080.

Următorul e un exemplu de secvenşă de întrerupere.



Exhipamentul 1 semnalează o întrerupere UCP-ului care e în execuţia instrucţiei 3COB. Execuţia acestei instrucţii e terminată. Numărătorul de program rămîne setat la 3COC, şi execută instrucţiunea RST OH furnizată de echipamentul 1.

Pînă cînd aceasta e o apelare la locaţia zero, 3COC e pusă în stivă şi controlul programului e transferat la locaţia 0000H. (Această subrutină poate realiza salturi, apelări, deci orice alte operaţii). Cînd s-a executat RETURN, adresa 3COC e scoasă (popped) din stivă şi repusă în numărătorul de program continuîndu-se execuţia.

Subrutine de intrerupere scrise

In general, fiecare bit de registru sau de condiţie schimbat de o subrutină de întrerupere trebuie restocat înainte de revenirea la programul întrerupt, astfel intervin erori.

De exemplu, presupunem un program întrerupt înainte de instrucţiuhea JC LOC şi bitul de transport este 1. Dacă subrutina de întrerupere se îmtîmplă să reseteze bitul de transport

înainte de revenirea la programul întrerupt, nu va avea loc saltul la LOC, iar programul întrerupt va produce rezultate erenate. Ca orice altă subrutină, orice rutină de întrerupere trebuie să salveze biţii de condiţie şi să îi restocheze înainte realizării operaţiei RETURN. (Soluţia cea mai convenabilă e de a salva datele în stivă prin operaţia PUSH sau POP).

Mai mult decît atît, sistemul de întreruperi e inhibat automat cînd se recunoaşte o întrerupere. Excepţie fac cazuri speciale, o subrutină de întrerupere trebuie să includă o instrucţie EI pentru a permite detecţia şi deservirea unei viitoare întreruperi, O instrucţiune putînd fi executată fiind după EI, subrutina de întrerupere poate fo ea însăşi întreruptă. Acest proces poate continua pînă la orice nivel, dar atît timp cît datele utile sînt salvate şi restocate, se continuă execuţia corectă a programului, în mod automat.

O subrutină tipică de întrerupere, poate apare ca mai jos :

```
Cod     Operand     Comentariu
PUSH     PSW
EI
.
.
.
.
.
POP      PSW
RET
```

Anexa A. Sumarul instrucţiunilor

Această anexă prezintă instrucţiunile şi stările asociate fiecărei instrucţiuni UCP 8080. Instrucţiunile sînt listate cu mnemonic şi cod de operaţie.

DDD - registrul destinaţie ; SSS - registrul sursă. Intreruperea registrelor.

DDD sau SSS	
000	Registrul D
001	Regįstrul C
010	Registrul D
011	Registrul E
100	Registrul H
101	Registrul L
110	Registrul
111	

Timpul de execuţie al instrucţiunii = numărul perioadelor înmulţit cu durata perioadelor.

O perioadă de timp variază între 480 nanosecunde la 2 microsecunde la 8080 şi de la 320 nanosecunde la 2 microsecunde la 8085. Cînd se simbolizează două numere pentru o perioadă de timp, de exemplu 5/11, perioada mai scurtă e valabilă dacă nu s-a pus o anume condiţie, iar cea mai lungă dată s-a pus.

Mnemonic	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_o	Număr de prioade de timp	
									8080	8085
CALL	1	1	0	0	1	1	0	1	17	18
CC	1	1	0	1	1	1	0	0	11/17	9/18
CNC	1	1	0	1	0	1	0	0	11/17	9/18
CZ	1	1	0	0	1	1	0	0	11/17	9/18
CNZ	1	1	0	0	0	1	0	0	11/17	9/18
CP	1	1	1	1	0	1	0	0	11/17	9/18
CM	1	1	1	1	1	1	0	0	11/17	9/18
CPE	1	1	1	0	1	1	0	0	11/17	9/17
CPO	1	1	1	0	0	1	0	0	11/17	9/18
RET	1	1	0	0	1	0	0	1	10	10
RC	1	1	0	1	1	0	0	0	5/11	6/12
RNC	0	1	0	1	0	0	0	0	5/11	6/12
RZ	1	1	0	0	1	0	0	0	5/11	6/12

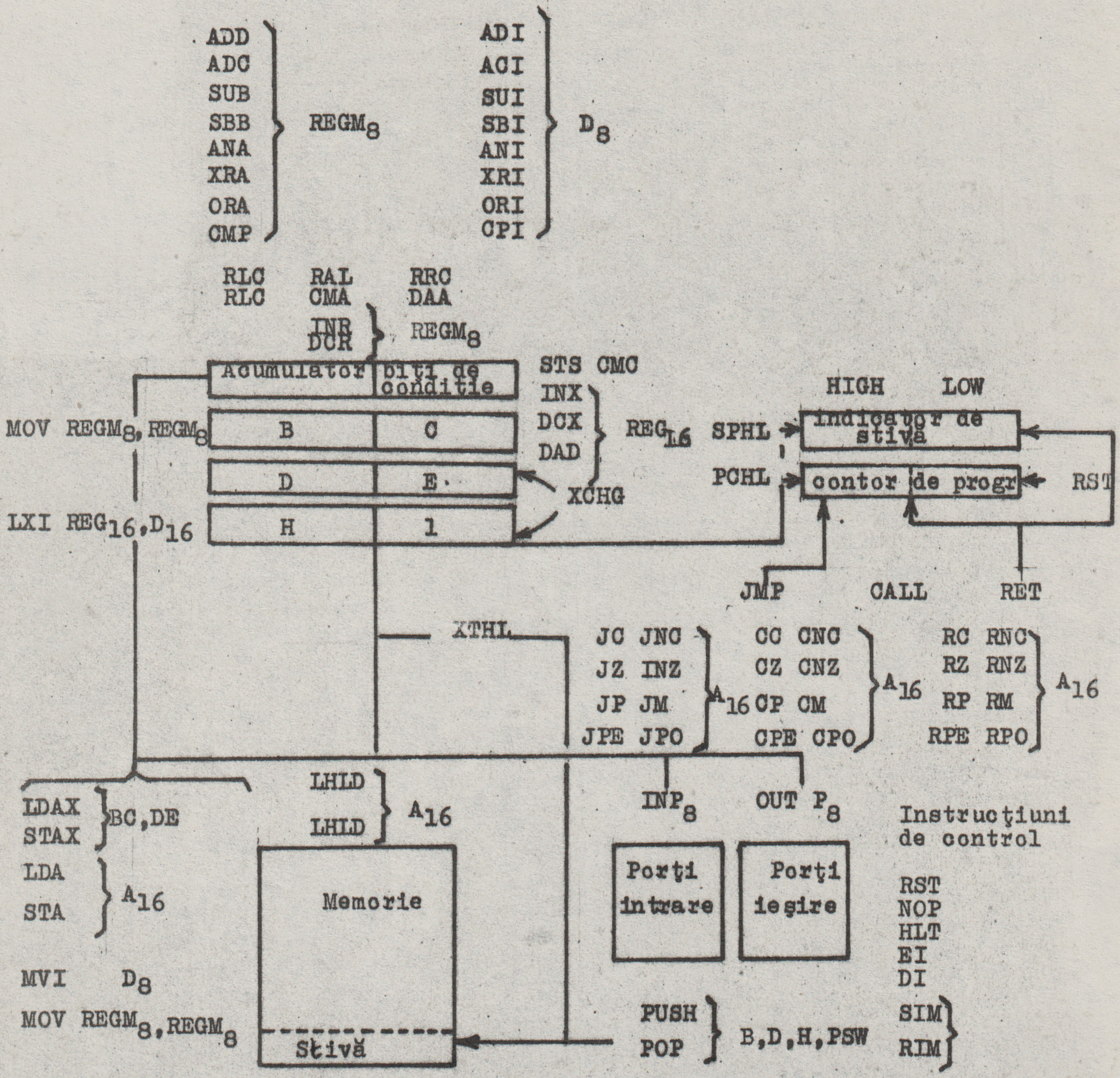
MNEMONIC	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Număr de perioade de timp	
									8080	8085
RNZ	1	1	0	0	0	0	0	0	5/11	6/12
RP	1	1	1	1	0	0	0	0	5/11	6/12
RM	1	1	1	1	1	0	0	0	5/11	6/12
RPE	1	1	1	0	1	0	0	0	5/11	6/12
RPO	1	1	1	0	0	0	0	0	5/11	6/12
RST	1	1	A	A	A	1	1	1	11	12
IN	1	1	0	1	1	0	1	1	10	10
OUT	1	1	0	1	0	0	1	1	10	10
LXI B	0	0	0	0	0	0	0	1	10	10
LXI D	0	0	0	1	0	0	0	1	10	10
LXI H	0	0	1	0	0	0	0	1	10	10
LXI SP	0	0	1	1	0	0	0	1	10	10
PUSH B	1	1	0	0	0	1	0	1	11	12
PUSH D	1	1	0	1	0	1	0	1	11	12
PUSH H	1	1	1	0	0	1	0	1	11	12
PUSH PSW	1	1	1	1	0	1	0	1	11	12
POP B	1	1	0	0	0	0	0	1	10	10
POP D	1	1	0	1	0	0	0	1	10	10
POP H	1	1	1	0	0	0	0	1	10	10
POP PSW	1	1	1	1	0	0	0	1	10	10
STA	0	0	1	1	0	0	1	0	13	13
LDA	0	0	1	1	1	0	1	0	13	13
XCHG	1	1	1	0	1	0	1	1	4	4
XTHL	1	1	1	0	0	0	1	1	18	16
SPHL	1	1	1	1	1	0	0	1	5	6
PCHL	1	1	1	0	1	0	0	1	5	6
DAD B	0	0	0	0	1	0	0	1	10	10
DAD D	0	0	0	1	1	0	0	1	10	10
DAD H	0	0	1	0	1	0	0	1	10	10
DAD SP	0	0	1	1	1	0	0	1	10	10
STAX B	0	0	0	0	0	0	1	0	7	7
STAX D	0	0	0	1	0	0	1	0	7	7
LDAX B	0	0	0	0	1	0	1	0	7	7
LDAX D	0	0	0	1	1	0	1	0	7	7
INX B	0	0	0	0	0	0	1	1	5	6
INX D	0	0	0	1	0	0	1	1	5	6
INX H	0	0	1	0	0	0	1	1	5	6
INX SP	0	0	1	1	0	0	1	1	5	6
MOV r_1,r_2	0	1	D	D	D	S	S	S	5	4
MOV M,r	0	1	1	1	0	S	S	S	7	7
MOV r,M	0	1	D	D	D	1	1	0	7	7
HLT	0	1	1	1	0	1	1	0	7	5
MVI r	0	0	D	D	D	1	1	0	7	7
MVI M	0	0	1	1	0	1	1	0	10	10
INR	0	0	D	D	D	1	0	0	5	4
DCR	0	0	D	D	D	1	0	1	5	4

MNEMONIC	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Număr de perioade de timp	
									8080	8085
INR A	0	0	1	1	1	1	0	0	5	4
DCR A	0	0	1	1	1	1	0	1	5	4
INR M	0	0	1	1	0	1	0	0	10	10
DCR M	0	0	1	1	0	1	0	1	10	10
ADD r	1	0	0	0	0	S	S	S	4	4
ADC r	1	0	0	0	1	S	S	S	4	4
SUB r	1	0	0	1	0	S	S	S	4	4
SBB r	1	0	0	1	1	S	S	S	4	4
AND r	1	0	1	0	0	S	S	S	4	4
XRA r	1	0	1	0	1	S	S	S	4	4
ORA r	1	0	1	1	0	S	S	S	4	4
CMP r	1	0	1	1	1	S	S	S	4	4
ADD M	1	0	0	0	0	1	1	0	7	7
ADC M	1	0	0	0	1	1	1	0	7	7
SUB M	1	0	0	1	0	1	1	0	7	7
SBB M	1	0	0	1	1	1	1	0	7	7
AND M	1	0	1	0	0	1	1	0	7	7
XRA M	1	0	1	0	1	1	1	0	7	7
ORA M	1	0	1	1	0	1	1	0	7	7
CMP M	1	0	1	1	1	1	1	0	7	7
ADI	1	1	0	0	0	1	1	0	7	7
ACI	1	1	0	0	1	1	1	0	7	7
SUI	1	1	0	1	0	1	1	0	7	7
SBI	1	1	0	1	1	1	1	0	7	7
ANI	1	1	1	0	0	1	1	0	7	7
XRI	1	1	1	0	1	1	1	0	7	7
ORI	1	1	1	1	0	1	1	0	7	7
CPI	1	1	1	1	1	1	1	0	7	7
RLC	0	0	0	0	0	1	1	1	4	4
RRC	0	0	0	0	1	1	1	1	4	4
RAL	0	0	0	1	0	1	1	1	4	4
RAR	0	0	0	1	1	1	1	1	4	4
JMP	1	1	0	0	0	0	1	1	10	10
JC	1	1	0	1	1	0	1	0	10	7/10
JNC	1	1	0	1	0	0	1	0	10	7/10
JZ	1	1	0	0	1	0	1	0	10	7/10
JNZ	1	1	0	0	0	0	1	0	10	7/10
JP	1	1	1	1	0	0	1	0	10	7/10
JM	1	1	1	1	1	0	1	0	10	7/10
JPE	1	1	1	0	1	0	1	0	10	7/10
JPO	1	1	1	0	0	0	1	0	10	7/10
DCX B	0	0	0	0	1	0	1	1	5	6
DCX D	0	0	0	1	1	0	1	1	5	6
DCX H	0	0	1	0	1	0	1	1	5	6
DCX SP	0	0	1	1	1	0	1	1	5	6

MNEMONIC	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Număr de perioade de timp	
									8080	8085
CMA	0	0	1	0	1	1	1	1	4	4
STC	0	0	1	1	0	1	1	1	4	4
CMC	0	0	1	1	1	1	1	1	4	4
DAA	0	0	1	0	0	1	1	1	4	4
SHLD	0	0	1	0	0	0	1	0	16	16
LHLD	0	0	1	0	1	0	1	0	16	16
RIM	0	0	1	0	0	0	0	0	–	4
SIM	0	0	1	1	0	0	0	0	–	4
EI	1	1	1	1	1	0	1	1	4	4.
DI	1	1	1	1	0	0	1	1	4	4
NOP	0	0	0	0	0	0	0	0	4	4

Instrucţiunile unităţii centrale de prelucrare în secvenţe de cod de operare:

OP CODE	MNEMONIC	OP CODE	MNEMONIC	OP CODE	MNEMONIC	OP CODE	MNEMONIC	OP CODE	MNEMONIC	OP CODE	MNEMONIC
00	NOP	2B	DCX H	56	MOV D,M	81	ADD C	AC	XRA H	D7	RST 2
01	LXI B,D16	2C	INR L	57	MOV D,A	82	ADD D	AD	XRA L	D8	RC
02	STAX B	2D	DCR L	58	MOV E,B	83	ADD E	AE	XRA M	D9	—
03	INX B	2E	MVI L,D8	59	MOV E,C	84	ADD H	AF	XRA A	DA	JC Adr
04	INR B	2F	CMA	5A	MOV E,D	85	ADD L	B0	ORA B	DB	IN D8
05	DCR B	30	SIM	5B	MOV E,E	86	ADD M	B1	ORA C	DC	CC Adr
06	MVI B,D8	31	LXI SPD16	5C	MOV E,H	87	ADD A	B2	ORA D	DD	—
07	RLC	32	STA Adr	5D	MOV E,L	88	ADC B	B3	ORA E	DE	SBI D8
08	—	33	INX SP	5E	MOV E,M	89	ADC C	B4	ORA H	DF	RST 3
09	DAD B	34	INR M	5F	MOV E,A	8A	ADC D	B5	ORA L	E0	RPO
0A	LDAXB	35	DCR M	60	MOV H,B	8B	ADC E	B6	ORA M	E1	POP H
0B	DCX B	36	MVI M,D8	61	MOV H,C	8C	ADC H	B7	ORA A	E2	JPO Adr
0C	INR C	37	STC	62	MOV H,D	8D	ADC L	B8	CMP B	E3	XTHL
0D	DCR C	38	--	63	MOV H,E	8E	ADC M	B9	CMP C	E4	CPO Adr
0E	MVI C,D8	39	DAD SP	64	MOV H,H	8F	ADC A	BA	CMP D	E5	PUSH H
0F	RRC	3A	LDA Adr	65	MOV H,L	90	SUB B	BB	CMP E	E6	ANI D8
10	--	3B	DCX SP	66	MOV H,M	91	SUB C	BC	CMP H	E7	RST 4
11	LXI D,D16	3C	INR A	67	MOV H,A	92	SUB D	BD	CMP L	E8	RPE
12	STAX D	3D	DCR A	68	MOV L,B	93	SUB E	BE	CMP M	E9	PCHL
13	INX D	3E	MVI A,D8	69	MOV L,C	94	SUB H	BF	CMP A	EA	JPE Adr
14	INR D	3F	CMC	6A	MOV L,D	95	SUB L	C0	RNZ	EB	XCHG
15	DCR D	40	MOV B,B	6B	MOV L,E	96	SUB M	C1	POP B	EC	CPE Adr
16	MVI D,D8	41	MOV B,C	6C	MOV L,H	97	SUB A	C2	JNZ Adr	ED	--
17	RAL	42	MOV B,D	6D	MOV L,L	98	SBB B	C3	JMP Adr	EE	XRI D8
18	--	43	MOV B,E	6E	MOV L,M	99	SBB C	C4	CNZ Adr	EF	RST 5
19	DAD D	44	MOV B,H	6F	MOV L,A	9A	SBB D	C5	PUSH B	F0	RP
1A	LDAXD	45	MOV B,L	70	MOV M,B	9B	SBB E	C6	ADI D8	F1	POP PSW
1B	DCX D	46	MOV B,M	71	MOV M,C	9C	SBB H	C7	RST 0	F2	JP Adr
1C	INR E	47	MOV B,A	72	MOV M,D	9D	SBB L	C8	RZ	F3	DI
1D	DRC E	48	MOV C,B	73	MOV M,E	9E	SBB M	C9	RET Adr	F4	CP Adr
1E	MVI E,D8	49	MOV C,C	74	MOV M,H	9F	SBB A	CA	JZ	F5	PUSH PSW
1F	RAR	4A	MOV C,D	75	MOV M,L	A0	ANA B	CB	--	F6	ORI D8
20	RIM	4B	MOV C,E	76	HLT	A1	ANA C	CC	CZ Adr	F7	RST 6
21	LXI H,D16	4C	MOV C,H	77	MOV M,A	A2	ANA D	CD	CALL Adr	F8	RM
22	SHLD Adr	4D	MOV C,L	78	MOV A,B	A3	ANA E	CE	ACI D8	F9	SPHL
23	INX H	4E	MOV C,M	79	MOV A,C	A4	ANA H	CF	RST 1	FA	JM Adr
24	INR H	4F	MOV C,A	7A	MOV A,D	A5	ANA L	D0	RNC	FB	EI
25	DCR H	50	MOV D,B	7B	MOV A,E	A6	ANA M	D1	POP D	FC	CM Adr
26	MVI H,D8	51	MOV D,C	7C	MOV A,H	A7	ANA A	D2	JNC Adr	FD	--
27	DAA	52	MOV D,D	7D	MOV A,L	A8	XRA B	D3	OUT D8	FE	CPI D8
28	--	53	MOV D,E	7E	MOV A,M	A9	XRA C	D4	CNC Adr	FF	RST 7
29	DAD H	54	MOV D,H	7F	MOV A,A	AA	XRA D	D5	PUSH D		
2A	LHLD Adr	55	MOV D,L	80	ADD B	AB	XRA E	D6	SUI D8		



$REGM_8$ - Operandul poate specifica unul din registrele de 8 biţi,A,B,C,D,E,H. L sau M (registrul de memorie M este specificat de adresa de 16 biţi continuată în registrul pereche HL).
Instrucţiunea MOV care foloseşte doi operanzi poate specifica un registru de memorie M, numai pentru unul din operanzi.

D_8 - Defineşte un operand imediat de 8 biţi

A_{16} - Defineşte o adresă de 16 biţi

P_8 - Defineşte un număr de 8 biţi,numărul unui port

REG_{16} - Defineşte un registru pereche de 16 biţi (BC,DE,HL sau SP)

D_{16} - Defineşte un operand imediat de 16 biţi

Anexa B. Sumarul instrucţiunilor asamblorului

Notaţii :

Termeni	Interpretare
Expresie	Expresie numerică evaluată pe parcursul asamblării, are 8 sau 16 biţi, funcţie de provenienţa instrucţiunii.
Listă	Serie de valori simbolice sau expresii separate prin virgulă.
Nume	Nume simbolic terminat prin blanc.
Nul	Cîmpul trebuie să fie gol, sau rezultă eroare.
Oplab	Etichetă opţională, terminată prin două puncte.
Parametru	Parametrii fictivi sînt simboluri ocupînd locul parametrilor reali (valori simbolice sau expresii) specificaţi în altă parte în program.
Linie, şir	Seria de orice caractere ASCII închise de ghilimele simple; ghilimele simple cu un rînd nul între ele, arată ca două ghilimele simple consecvente
Text	Serie de caractere ASCII

Caracter	Funcţie
&	Legătură pentru conectarea simbolurilor .
;;	Punct şi virgulă dublată, în faţa comentariului din definiţia macro şi pentru a preveni includerea comentariului în macro expansiune.
<>	Paramteză ascuţită; delimitează textele cum ar fi, listele, care la rîndul lor pot conţine alţi delimitatori.
!	Semnul exclamării (caracter de scăpare). Plasat în faţa unui delimitator care trebuie trecut ca literal într-un parametru real. Pentru a trece semnul exclamării literal, se scrie '!!'.
%	Precedă parametrii reali ce urmează a fi evaluaţi imediat, cînd se apelează o macro.

Sumarul Pseudoinstrucţiunilor

Erichetă	Mnemonic	Operand	Funcţie
oplab:	DB	expresie sau şir	Defineşte byte-ul (8 biţi) de date.Expresia trebuie să fie de un byte.
oplab:	DS	expresie	Rezervă domeniul datelor memorate de lungimea specificată
oplab:	DW	expresie sau şir	Defineşte cuvinte de date de 16 biţi. Siruri limitate la 1-2 caractere.
oplab:	ELSE	nul	Asamblare condiţionată.Codurile între ELSE şi ENDIF sînt asamblate dacă expresia în clauză IF e FALSE.
oplab:	END	expresie	Terminarea rulării asamblorului.Trebuie să fie ultima afirmaţie a programului. Execuţia programului începe la locaţia indicată de expresie, cînd o prezintă,cînd nu,începe la locaţia 0.
oplab:	ENDIF	nul	Termină blocul de asamblare condiţională.
nume:	EQU	expresie	Defineşte 'numele' simbolic (simbolul) cu valoarea 'expresiei'. Simbolul nu este redefinibil.
oplab:	IF	expresie	Asamblează codul între IF şi următorul ELSE sau ENDIF,dacă expresia e adecvată
oplab:	ORG	expresie	Aşează numărătorul de adresă la 'expresion'
nume	SET	expresie	Defineşte 'numele' simbolic cu valoarea 'expresiei'. Simbolul poate fi redefinit.

Pseudoinstrucţiuni macro

Format			Funcţie
Eticheta	Mnemonic	Operand	
nul	ENDM	nul	Termină definiţia macro
oplab:	EXTIM	nul	Alternează terminatorul definiţiei macro (vezi ENDM)
oplab:	IRP	parametru fictiv, list	Repetă secvenţa de instrucţiune substituind un caracter din lista de parametrii fictivi în fiecare iteraţie.
oplab:	IRPC	parametru fictiv, text	Repetă secvenţa de instrucţiune substituind un caracter din 'textul' cu parametrii fictivi' în fiecare interaţie
nul	LOCAL	numele etichetei	Specifică eticheta în definiţia macro pentru a vaea efect local
nume	MACRO	parametru fictiv	Defineşte (numele) macro şi parametrii fictivi pentru a fi folosiţi în definiţia macro.
oplab:	REPT	expresie	Repetă blocul REPT de 'expresie' ori

Instrucţiuni de relocatare

Format			Funcţie
Etichetă	Mnemonic	Operand	
oplab:	ASEG	nul	Asamblează instrucţiuni subsecvente şi date,în modul abdolut.
oplab:	CSEG	specificaţii ale limitărilor	Asamblează instrucţiuni şi date în modul relocabil,folosind numărul de adrése de instrucţiuni.

Etichetă	Mnemonic	Operand	
oplab:	DSEG	specificații ale limitelor	Asamblează instrucțiunile subsecvente și datele în modul realocabil folosind numărătorul de adrese de date
oplab:	EXTRN	numecc	Idenfifică simboluri în acest modul de program dat, definite în alt modul.
oplab:	PUBLIC	nume	Identifică simboluri definite în acest modul care vor fi accesibile altor module.
oplab:	NUME	numele modulului	Asignează un nume modulului de program.
oplab:	STKLN	expresie	Specifică numărul de byte-ți care trebuiesc rezervați pentru stivă, pentru acest modul.

Anexa C. SETUL DE CARACTERE A.S.C.I.I.

CODURI ASCII

Semne grafice sau comenzi	ASCII (HEXADECIMAL)
NUL	00
SOH	01
STX	02
ETX	03
EOT	04
ENQ	05
ACK	06
BEL	07
BS	08
HT	09
LF	0A
VT	0B
FF	0C
CR	0D
SO	0E
SI	0F
DLE	10
DC1 (X-ON)	11
DC2 (TAPE)	12
DC3 (X-OFF)	13
DC4 (~~TAPE~~)	14
N/	15
SYN	16
ETB	17
CAN	18
EM	19
SUB	1A
ESC	1B
FS	1C
GS	1D
RS	1E
US	1F
SP	20
!	21
"	22
#	23
$	24
%	25
&	26
'	27
(	28
)	29
*	2A

Semne grafice sau comenzi	ASCII (HEXADECIMAL)
+	2B
,	2C
–	2D
.	2E
/	2F
0	30
1	31
2	32
3	33
4	34
5	35
6	36
7	37
8	38
9	39
:	3A
;	3B
<	3C
=	3D
>	3E
?	3F
@	40
A	41
B	42
C	43
D	44
E	45
F	46
G	47
H	48
I	49
J	4A
K	4B
L	4C
M	4D
N	4E
O	4F
P	50
Q	51
R	52
S	53
T	54
U	55

Semne grafice sau comenzi	ASCII (HEXADECIMAL)
V	56
W	57
X	58
Y	59
Z	5A
[	5B
\	5C
]	5D
∧ (↑)	5E
– (←)	5F
‘	60
a	61
b	62
c	63
d	64
e	65
f	66
g	67
h	68
i	69
j	6A
k	6B
l	6C
m	6D
n	6E
o	6F
p	70
q	71
r	72
s	73
t	74
u	75
v	76
w	77
x	78
y	79
z	7A
{	7B
|	7C
} (ALT MODE)	7D
	7E
DEL (RUB OUT)	7F

ANEXA D

TABELE CU CONVERSIE BINAR-ZECIMAL-HEXAZECIMAL

lui Puterile lui 2 (în baza 10)

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5
2 199 023 255 552	41	0.000 000 000 000 454 747 350 886 464 118 957 519 531 25
4 398 046 511 104	42	0.000 000 000 000 227 373 675 443 232 059 478 759 765 625
8 796 093 022 208	43	0.000 000 000 000 113 686 837 721 616 029 739 379 882 812 5
17 592 186 044 416	44	0.000 000 000 000 056 843 418 860 808 014 869 689 941 406 25
35 184 372 088 832	45	0.000 000 000 000 028 421 709 430 404 007 434 844 970 703 125
70 368 744 177 664	46	0.000 000 000 000 014 210 854 715 202 003 717 422 485 351 562 5
140 737 488 355 328	47	0.000 000 000 000 007 105 427 357 601 001 858 711 242 675 781 25
281 474 976 710 656	48	0.000 000 000 000 003 552 713 678 800 500 929 355 621 337 890 625
562 949 953 421 312	49	0.000 000 000 000 001 776 356 839 400 250 464 677 810 668 945 312 5
1 125 899 906 842 624	50	0.000 000 000 000 000 888 178 419 700 125 232 338 905 334 472 656 25
2 251 799 813 685 248	51	0.000 000 000 000 000 444 089 209 850 062 616 169 452 667 236 328 125
4 503 599 627 370 496	52	0.000 000 000 000 000 222 044 604 925 031 308 084 726 333 618 164 062 5
9 007 199 254 740 992	53	0.000 000 000 000 000 111 022 302 462 515 654 042 363 166 809 082 031 25
18 014 398 509 481 984	54	0.000 000 000 000 000 055 511 151 231 257 827 021 181 583 404 541 015 625
36 028 797 018 963 968	55	0.000 000 000 000 000 027 755 575 615 628 913 510 590 791 702 270 507 812 5
72 057 594 037 927 936	56	0.000 000 000 000 000 013 877 787 807 814 456 755 295 395 851 135 253 906 25
144 115 188 075 855 872	57	0.000 000 000 000 000 006 938 893 903 907 228 377 647 697 925 567 676 950 125
288 230 376 151 711 744	58	0.000 000 000 000 000 003 469 446 951 953 614 188 823 848 962 783 813 476 562 5
576 460 752 303 423 488	59	0.000 000 000 000 000 001 734 723 475 976 807 094 411 924 481 391 906 738 281 25
1 152 921 504 606 846 976	60	0.000 000 000 000 000 000 867 361 737 988 403 547 205 962 240 695 953 369 140 625
2 305 843 009 213 693 952	61	0.000 000 000 000 000 000 433 680 868 994 201 773 602 981 120 347 976 684 570 312 5
4 611 686 018 427 387 904	62	0.000 000 000 000 000 000 216 840 434 497 100 886 801 490 560 173 988 342 285 156 25
9 223 372 036 854 775 808	63	0.000 000 000 000 000 000 108 420 217 248 550 443 400 745 280 086 994 171 142 578 125

Puteri ale lui 16 (în baza 10)

16^n	n	16^{-n}				
1	0	0.10000	00000	00000	00000	x 10
16	1	0.62500	00000	00000	00000	x 10^{-1}
256	2	0.39062	50000	00000	00000	x 10^{-2}
4 096	3	0.24414	06250	00000	00000	x 10^{-3}
65 536	4	0.15258	78906	25000	00000	x 10^{-4}
1 048 576	5	0.95367	43164	06250	00000	x 10^{-6}
16 777 216	6	0.59604	64477	53906	25000	x 10^{-7}
268 435 456	7	0.37252	90298	46191	40625	x 10^{-8}
4 294 967 296	8	0.23283	06436	53869	62891	x 10^{-9}
68 719 476 736	9	0.14551	91522	83668	51807	x 10^{-10}
1 099 511 627 776	10	0.90949	47017	72928	23792	x 10^{-12}
17 592 186 044 416	11	0.56843	41886	08080	14870	x 10^{-13}
281 474 976 710 656	12	0.35527	13678	80050	09294	x 10^{-14}
4 503 599 627 370 496	13	0.22204	46049	25031	30808	x 10^{-15}
72 057 594 037 927 936	14	0.13877	78780	78144	56755	x 10^{-16}
1 152 921 504 606 846 976	15	0.86736	17379	88403	54721	x 10^{-18}

Puteri ale lui 10 (în baza 16)

10^n	n	10^{-n}				
1	0	1.0000	0000	0000	0000	
A	1	0.1999	9999	9999	999A	
64	2	0.28F5	C28F	5C28	F5C3	x 16^{-1}
3E8	3	0.4189	374B	C6A7	EF9E	x 16^{-2}
2710	4	0.68DB	8BAC	710C	B296	x 16^{-3}
1 86A0	5	0.A7C5	AC47	1B47	8423	x 16^{-4}
F 4240	6	0.10C6	F7A0	B5ED	8D37	x 16^{-4}
98 9680	7	0.1AD7	F29A	BCAF	4858	x 16^{-5}
5F5 E100	8	0.2AF3	1DC4	6118	73BF	x 16^{-6}
3B9A CA00	9	0.44B8	2FA0	9B5A	52CC	x 16^{-7}
2 540B E400	10	0.6DF3	7F67	5EF6	EADF	x 16^{-8}
17 4876 E800	11	0.AFEB	FF0B	CB24	AAFF	x 16^{-9}
E8 D4A5 1000	12	0.1197	9981	2DEA	1119	x 16^{-9}
918 4E72 A000	13	0.1C25	C268	4976	81C2	x 16^{-10}
5AF3 107A 4000	14	0.2D09	370D	4257	3604	x 16^{-11}
3 8D7E A4C6 8000	15	0.480E	BE7B	9D58	566D	x 16^{-12}
23 8652 6FC1 0000	16	0.734A	CA5F	6226	F0AE	x 16^{-13}
163 4578 5D8A 0000	17	0.B877	AA32	36A4	B449	x 16^{-14}
DE0 B6B3 A764 0000	18	0.1272	5DD1	D243	ABA1	x 16^{-14}
8AC7 2304 89E8 0000	19	0.1D83	C94F	B6D2	AC35	x 16^{-15}

CONVERSIA NUMERELOR INTREGI DIN HEXAZECIMAL IN ZECIMAL

Pentru conversia unor numere întregi mai mari se poate utiliza următorul tabel :

Hexadecimal	Decimal	Hexadecimal	Decimal
01 000	4 096	20 000	131 072
02 000	8 192	30 000	196 608
03 000	12 288	40 000	262 144
04 000	16 384	50 000	327 680
05 000	20 480	60 000	393 216
06 000	24 576	70 000	458 752
07 000	28 672	80 000	524 288
08 000	32 768	90 000	589 824
09 000	36 864	A0 000	655 360
0A 000	40 960	B0 000	720 896
0B 000	45 056	C0 000	786 432
0C 000	49 152	D0 000	851 968
0D 000	53 248	E0 000	917 504
0E 000	57 344	F0 000	983 040
0F 000	61 440	100 000	1 048 576
10 000	65 536	200 000	2 097 152
11 000	69 632	300 000	3 145 728
12 000	73 728	400 000	4 194 304
13 000	77 824	500 000	5 242 880
14 000	81 920	600 000	6 291 456
15 000	86 016	700 000	7 340 032
16 000	90 112	800 000	8 388 608
17 000	94 208	900 000	9 437 184
18 000	98 304	A00 000	10 485 760
19 000	102 400	B00 000	11 534 336
1A 000	106 496	C00 000	12 582 912
1B 000	110 592	D00 000	13 631 488
1C 000	114 688	E00 000	14 680 064
1D 000	118 784	F00 000	15 728 640
1E 000	122 880	1 000 000	16 777 216
1F 000	126 976	2 000 000	33 554 432

Tabelul următor conţine numere hexazecimale
între 0 - FFF şi corespondentele zecimale
între 0 - 4095

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
010	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
020	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
030	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
040	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
050	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
060	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
070	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
080	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
090	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A0	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B0	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C0	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D0	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E0	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F0	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255

D-4

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
100	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
110	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
120	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
130	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
140	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0331	0333	0334	0335
150	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
160	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
170	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
180	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
190	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A0	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B0	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C0	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D0	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E0	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F0	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511
200	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
210	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
220	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
230	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
240	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
250	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
260	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
270	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
280	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
290	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A0	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B0	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C0	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D0	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E0	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F0	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767
300	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
310	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
320	0800	0301	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
330	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
340	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
350	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
360	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
370	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
380	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
390	0212	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A0	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B0	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0056	0957	0958	0959
3C0	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0872	0973	0974	0975
3D0	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E0	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F0	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
400	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
410	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
420	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
430	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
440	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
450	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
460	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
470	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
480	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
490	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A0	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B0	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C0	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D0	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E0	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F0	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
500	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
510	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
520	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
530	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
540	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
550	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
560	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
570	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
580	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
590	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A0	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B0	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C0	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D0	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E0	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F0	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535
600	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
610	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
620	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
630	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
640	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
650	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
660	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
670	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663
680	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
690	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A0	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B0	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C0	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D0	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E0	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F0	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
700	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
710	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
720	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
730	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
740	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
750	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
760	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
770	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
780	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
790	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A0	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B0	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C0	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D0	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E0	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F0	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047
800	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
810	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
820	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
830	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
840	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
850	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
860	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
870	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
880	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
890	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A0	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B0	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C0	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D0	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E0	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F0	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303
900	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
910	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
920	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
930	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
940	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
950	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
960	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
970	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
980	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
990	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A0	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B0	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C0	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D0	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E0	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F0	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A00	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A10	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A20	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A30	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A40	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A50	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A60	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A70	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A80	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A90	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA0	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB0	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC0	2752	2753	2754	2755	2756	2757	2758	2759	2760	4761	2762	2763	2764	2765	2766	2767
AD0	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE0	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF0	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
B00	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B10	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B20	2848	2849	2850	3851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B30	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B40	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B50	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B60	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B70	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B80	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B90	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA0	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB0	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC0	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD0	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE0	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF0	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071
C00	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C10	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C20	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C30	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C40	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C50	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C60	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C70	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C80	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C90	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA0	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB0	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC0	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD0	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE0	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF0	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
D00	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D10	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D20	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D30	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D40	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D50	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D60	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D70	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D80	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D90	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA0	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB0	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC0	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD0	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE0	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF0	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583
E00	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E10	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E20	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E30	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E40	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E50	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E60	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E70	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E80	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E90	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA0	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB0	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC0	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED0	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE0	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF0	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839
F00	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F10	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F20	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F30	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F40	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F50	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F60	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F70	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F80	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F90	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA0	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB0	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC0	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD0	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE0	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF0	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

Anexa E. Codificarea în hexazecimal a instrucţiunilor μp 8080

Operaţii cu acumulatorul

80 ADD B
81 ADD C
82 ADD D
83 ADD E
84 ADD H
85 ADD L
86 ADD M
87 ADD A
88 ADC B
89 ADC C
8A ADC D
8B ADC E
8C ADC H
8D ADC L
8E ADC M
8F ADC A

90 SUB B
91 SUB C
92 SUB D
93 SUB E
94 SUB H
95 SUB L
96 SUB M
97 SUB A
98 SBB B
99 SBB C
9A SBB D
9B SBB E
9C SBB H
9D SBB L
9E SBB M
9F SBB A

A0 ANA B
A1 ANA C
A2 ANA D
A3 ANA E
A4 ANA H
A5 ANA L
A6 ANA M
A7 ANA A
A8 XRA B
A9 XRA C
AA XRA D
AB XRA E
AC XRA H
AD XRA L
AE XRA M
AF XRA A

TRANSFERURI

40 MOV B,B
41 MOV B,C
42 MOV B,D
43 MOV B,E
44 MOV B,H
45 MOV B.L
46 MOV B,M
47 MOV B,A
48 MOV C,B
49 MOV C,C
4A MOV C,D
4B MOV C,E
4C MOV C,H
4D MOV C,L
4E MOV C,M
4F MOV C,A
50 MOV D,B
51 MOV D,C
52 MOV D,D
53 MOV D,E
54 MOV D,H
55 MOV D,L
56 MOV D,M
57 MOV D,A
58 MOV E,B
59 MOV E,C
5A MOV E,D
5B MOV E,E
5C MOV E,H
5D MOV E,L
5E MOV E,M
5F MOV E,A
60 MOV H,B
61 MOV H,C
62 MOV H,D
63 MOV H,E
64 MOV H,H
65 MOV H,L
66 MOV H.M
67 MOV H,A
68mMOV L,B
69 MOV L,C
6A MOV L,D
6B MOC L,E
6C MOV L,H
6D MOV L,L
6E MOV L,M
6F MOV L,A

ROTATII

07 RLC
0F RRC
17 RAL
1F RAR

INCARCARI/ MEMORARI

0A LDAX B
1A LDAX D
2A LHLD Adr
3A LDA Adr

02 STAX B
12 STAX D
22 SHLD Adr
32 STA Adr

OPERATII CU STIVA

C5 PUSH B
D5 PUSH D
E5 PUSH H
F5 PUSH PSW

C1 POP B
D1 POP D
E1 POP H
F1 POP PSW
E3 XTHL
F9 SPHL

SPECIALE

EB XCHG
27 DAA
2F CMA
37 STC
3F CMC

00 MOP
76 HLT
F3 DI
FB EI

OPERATII IMEDIATE CU ACUMULATORUL

C6 ADI
CE ACI
D6 SUI
DE SBI
E6 ANI
EE XRI
F6 ORI
FE CPI

} D_8

INCREMENTARI

04 INR B
0C INR C
14 INR D
1C INR E
24 INR H
2C INR L
34 INR M
3C INR A

03 INX B
13 INX D
23 INX H
33 INX SP

DECREMENTARI

05 DCR B
0D DCR C
15 DCR D
1D DCR E
25 DCR H
2D DCR L
35 DCR M
3D DCR A
0B DCX B
1B DCX D
2B DCX H
3B DCX SP

ADUNARI PE 16 BITI

09 DAD B
19 DAD D
29 DAD H
39 DAD SP

SALTURI

C3 JMP
C2 JNZ
CA JZ
D2 JNC
DA JC
E2 JPO
EA JPE
F2 JP
FA JM
E9 PCHL

} Adr

B0 ORA B
B1 ORA C
B2 ORA D
B3 ORA E
B4 ORA H
B5 ORA L
B6 ORA M
B7 ORA A
B8 CMP B
B9 CMP C
BA CMP D
BB CMP E
BC CMP H
BD CMP L
BE CMP M
BF CMP A

INTRARE/IESIRE

D3 OUT
DB IN

70 MOV M,B
71 MOV M,C
72 MOV M,D
73 MOV M,E
74 MOV M,H
75 MOV M,L
77 MOV M,A
78 MOV A,B
79 MOV A,C
7A MOV A,D
7B MOV A,E
7C MOV A,H
7D MOV A,L
7E MOV A,M
7F MOV A,A

TRANSFER IMEDIAT

06 MVI B,
0E MVI C,
16 MVI D,
1E MVI E,
26 MVI H,
2E MVI L,
36 MVI M,
3E MVI A,

} D_8

INCARCARI
IMEDIATE

01 LXI B,
11 LXI D,
21 LXI H,
31 LXI SP,

} D_{16}

APEL SUBRUTINE

CD CALL
C4 CNZ
CC CZ
D4 CNC
CC CC
E4 CPO
EC CPE
F4 CP
FC CM

} Adr

REVENIRI

C9 RET
C0 RNZ
C8 RZ
D0 RNC
D8 RC
E0 RPO
E8 RPE
F0 RP
F8 RM

RESTART

C7 RST 0
CF RST 1
D7 RST 2
DF RST 3
E7 RST 4
EF RST 5
F7 RST 6
FF RST 7

Notaţii :

Adr = adresă pe 16 biţi
D_8 = constantă pe 8 biţi
D_{16} = constantă pe 16 biţi